Robot Dynamics Essential Based on RSL Lecture Note

Shuhei Watanabe

March 9, 2026

0 Notation Reference

0.1 Notation Conventions

This summary uses the following notation conventions, mapped from the original robotics textbook notation:

Type	Robotics (RSL)	This Summary
Scalar	italic lowercase: m, θ	italic lowercase: m, θ
Vector	italic: r, ω	bold lowercase: $\boldsymbol{r}$, $\boldsymbol{\omega}$
Matrix	uppercase: C, M, J	bold uppercase: $\boldsymbol{R}$, $\boldsymbol{M}$, $\boldsymbol{J}$
Rotation matrix	C_{AB}	$\boldsymbol{R}_{AB}$
Transformation	T_{AB}	$\boldsymbol{T}_{AB}$
Inertia tensor	Θ	$\boldsymbol{\Theta}$ (bold uppercase)
Identity	I, E	$\boldsymbol{I}_n$ ($n \times n$ identity)
Transpose	A^T	$\boldsymbol{A}^\top$

0.2 Frame Notation (Robotics-Specific)

Robotics uses **coordinate frames** to express vectors and matrices.

- ${}_A\boldsymbol{r}$ — vector $\boldsymbol{r}$ expressed in the coordinates of frame A.
- $\boldsymbol{r}_{OP}$ — vector pointing from point O to point P.
- ${}_A\boldsymbol{r}_{OP}$ — vector from O to P, expressed in frame A.
- $\boldsymbol{R}_{AB}$ — rotation matrix that transforms vectors from frame B to frame A: ${}_A\boldsymbol{r} = \boldsymbol{R}_{AB}\,{}_B\boldsymbol{r}$.
- ${}_A\boldsymbol{\omega}_{IB}$ — angular velocity of body B relative to inertial frame I, expressed in frame A.

Note that common frames are I = inertial (world/fixed), and B = body-fixed, $0, 1, \ldots, n$ = link frames.

0.3 Key Operators

For $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^\top$, the **skew-symmetric (cross-product) matrix** is:

$$[\boldsymbol{\omega}]_\times = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \qquad \boldsymbol{a} \times \boldsymbol{b} = [\boldsymbol{a}]_\times\, \boldsymbol{b}, \tag{1}$$

and the following properties hold:

- Anti-symmetry: $[\boldsymbol{a}]_\times^\top = -[\boldsymbol{a}]_\times$
- Rotation: $[\boldsymbol{R}\boldsymbol{a}]_\times = \boldsymbol{R}\,[\boldsymbol{a}]_\times\,\boldsymbol{R}^\top$ (for $\boldsymbol{R} \in SO(3)$)
- Linearity: $[\alpha\boldsymbol{a} + \beta\boldsymbol{b}]_\times = \alpha\,[\boldsymbol{a}]_\times + \beta\,[\boldsymbol{b}]_\times$
- Rank: $\operatorname{rank}([\boldsymbol{a}]_\times) = 2$ for $\boldsymbol{a} \neq \boldsymbol{0}$ (null space = $\operatorname{span}(\boldsymbol{a})$)
- Eigenvalues: $0, +i\|\boldsymbol{a}\|, -i\|\boldsymbol{a}\|$ (purely imaginary, consistent with skew-symmetry)

Other common notations are:

- $\dot{(\cdot)}$: time derivative (e.g., $\dot{\boldsymbol{q}} = d\boldsymbol{q}/dt$)
- $\delta(\cdot)$: virtual displacement (infinitesimal, at fixed time)
- $\partial/\partial q_j$: partial derivative with respect to generalized coordinate q_j
- $\hat{(\cdot)}$: unit vector (e.g., $\hat{\boldsymbol{z}}$ is the z-axis unit vector)
- $\|\cdot\|$: Euclidean norm unless otherwise specified
- $\boldsymbol{A} \succ \boldsymbol{0}$: $\boldsymbol{A}$ is positive definite (all eigenvalues > 0)
- $\boldsymbol{A} \succeq \boldsymbol{0}$: $\boldsymbol{A}$ is positive semi-definite

1 Introduction

- **Robot dynamics**: studies how forces and torques cause motion of robotic systems composed of rigid bodies connected by joints.
- **Rigid body assumption**: Each link is a rigid body: distances between any two points on the body remain constant.
- **Degrees of freedom (DoF)**: A free rigid body in 3D has 6 DoF (3 translational + 3 rotational). A robot with n joints (each 1-DoF) on a fixed base has n DoF.
- **Generalized coordinates**: $\boldsymbol{q} \in \mathbb{R}^n$: a minimal set of n independent coordinates that fully describe the configuration of the system. For a serial robot with revolute joints, $\boldsymbol{q}$ is the vector of joint angles.
- **Configuration space**: $\mathcal{Q}$: the set of all possible configurations $\boldsymbol{q}$. For a robot with n revolute joints, $\mathcal{Q} \subseteq \mathbb{T}^n$ (product of circles), though often parameterized as $\boldsymbol{q} \in \mathbb{R}^n$ with joint limits.
- **Task space**: $\mathcal{X}$: the space of end-effector poses. For a 6-DoF manipulator in 3D: $\mathcal{X} = \mathbb{R}^3 \times SO(3)$ (position + orientation).
- **Workspace**: the set of positions reachable by the end-effector. The workspace boundary corresponds to kinematic singularities where the robot is fully extended or folded.

Two main problems:

- *Kinematics*: geometry of motion without considering forces. Given joint angles $\boldsymbol{q}$, find end-effector pose (forward kinematics) or vice versa (inverse kinematics).
- *Dynamics*: relationship between forces/torques and motion. Given torques $\boldsymbol{\tau}$, find accelerations $\ddot{\boldsymbol{q}}$ (forward dynamics) or given desired $\ddot{\boldsymbol{q}}$, find required $\boldsymbol{\tau}$ (inverse dynamics).

Key pipeline:

$$\text{Joint torques } \boldsymbol{\tau} \xrightarrow{\text{dynamics}} \text{Joint accelerations } \ddot{\boldsymbol{q}} \xrightarrow{\text{integration}} \boldsymbol{q} \xrightarrow{\text{forward kin.}} \text{End-effector pose } \boldsymbol{x}$$

The inverse problems (IK, inverse dynamics) go in the reverse direction and are typically harder (may have multiple solutions or none).

2 Kinematics

2.1 Position and Velocity

- **Position** of a point P relative to origin O expressed in frame A: ${}_A\boldsymbol{r}_{OP} \in \mathbb{R}^3$.
- **Velocity** of point P as seen from frame A: ${}_A\boldsymbol{v}_P = {}_A\dot{\boldsymbol{r}}_{OP} = \frac{{}^A d}{dt}\,{}_A\boldsymbol{r}_{OP}$, where $\frac{{}^A d}{dt}$ denotes the time derivative as observed in frame A.
- **Change of frame** for a position vector: ${}_A\boldsymbol{r}_{OP} = {}_A\boldsymbol{r}_{OQ} + \boldsymbol{R}_{AB}\,{}_B\boldsymbol{r}_{QP}$.
- **Chain rule for positions**: $\boldsymbol{r}_{AC} = \boldsymbol{r}_{AB} + \boldsymbol{r}_{BC}$ (all expressed in the same frame).

Important distinction: The time derivative of a vector depends on the frame of differentiation—a vector constant in one frame may be time-varying in another due to relative rotation, which is the fundamental source of Coriolis and centrifugal effects.

2.2 Rotation

2.2.1 Rotation Matrix

A **rotation matrix** $\boldsymbol{R}_{AB} \in \mathbb{R}^{3\times 3}$ transforms vectors from frame B coordinates to frame A coordinates: ${}_A\boldsymbol{r} = \boldsymbol{R}_{AB}\,{}_B\boldsymbol{r}$. The columns of $\boldsymbol{R}_{AB}$ are the unit vectors of frame B's axes expressed in frame A. The following properties hold for the Special Orthogonal group $SO(3)$:

$$\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{R}\,\boldsymbol{R}^\top = \boldsymbol{I}_3 \text{ (orthogonal)}, \quad \det(\boldsymbol{R}) = +1, \quad \boldsymbol{R}^{-1} = \boldsymbol{R}^\top, \quad \boldsymbol{R}_{AC} = \boldsymbol{R}_{AB}\,\boldsymbol{R}_{BC} \text{ (composition)} \tag{2}$$

2.2.2 Elementary Rotations

Rotations about the x, y, z-axes by angles ϕ, θ, ψ are:

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \boldsymbol{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3}$$

respectively.

2.2.3 Euler Angles (ZYX Convention)

Any rotation can be parameterized by three angles $\boldsymbol{\Phi} = [\phi, \theta, \psi]^\top$ (roll, pitch, yaw) via the ZYX convention $\boldsymbol{R}(\boldsymbol{\Phi}) = \boldsymbol{R}_z(\psi)\,\boldsymbol{R}_y(\theta)\,\boldsymbol{R}_x(\phi)$. The expanded **full ZYX rotation matrix** is:

$$\boldsymbol{R}(\psi, \theta, \phi) = \begin{bmatrix} c\psi\, c\theta & c\psi\, s\theta\, s\phi - s\psi\, c\phi & c\psi\, s\theta\, c\phi + s\psi\, s\phi \\ s\psi\, c\theta & s\psi\, s\theta\, s\phi + c\psi\, c\phi & s\psi\, s\theta\, c\phi - c\psi\, s\phi \\ -s\theta & c\theta\, s\phi & c\theta\, c\phi \end{bmatrix} \tag{4}$$

where $c\psi = \cos\psi$, $s\psi = \sin\psi$, etc. The **Euler angles** can be extracted from a rotation matrix $\boldsymbol{R} = [r_{ij}]$ via $\theta = -\arcsin(r_{31}), \phi = \operatorname{atan2}(r_{32}, r_{33}), \psi = \operatorname{atan2}(r_{21}, r_{11})$. These formulas fail when $\cos\theta = 0$ (gimbal lock) [1].

2.2.4 Rotation Vector / Angle-Axis

Any rotation can be described by an axis $\hat{\boldsymbol{n}}$ ($\|\hat{\boldsymbol{n}}\| = 1$) and angle θ: $\boldsymbol{\phi} = \theta\,\hat{\boldsymbol{n}} \in \mathbb{R}^3$. **Rodrigues' rotation formula** reconstructs $\boldsymbol{R}$ from $(\theta, \hat{\boldsymbol{n}})$:

$$\boldsymbol{R} = \boldsymbol{I}_3 + \sin\theta\, [\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)\, [\hat{\boldsymbol{n}}]_\times^2 \tag{5}$$

This parameterization avoids gimbal lock (but has a singularity at $\theta = 0$ and $\theta = 2\pi$ when extracting the axis). When extracting angle and axis from $\boldsymbol{R}$, the following is useful:

$$\theta = \arccos\left(\frac{\operatorname{tr}(\boldsymbol{R}) - 1}{2}\right), \quad \hat{\boldsymbol{n}} = \frac{1}{2\sin\theta} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \tag{6}$$

Rodrigues' formula can be written as the **matrix exponential**:

$$\boldsymbol{R} = \exp([\boldsymbol{\phi}]_\times) = \exp(\theta\, [\hat{\boldsymbol{n}}]_\times) = \sum_{k=0}^{\infty} \frac{1}{k!} (\theta\, [\hat{\boldsymbol{n}}]_\times)^k \tag{7}$$

The series terminates due to the Cayley–Hamilton theorem ($[\hat{\boldsymbol{n}}]_\times^3 = -[\hat{\boldsymbol{n}}]_\times$). The **logarithmic map** is the inverse: $[\boldsymbol{\phi}]_\times = \log(\boldsymbol{R})$, extracting the rotation vector from a rotation matrix.

2.2.5 Angular Velocity

The **angular velocity** $\boldsymbol{\omega}_{AB}$ describes the rate of rotation of frame B relative to frame A. Relationship to the rotation matrix derivative:

$$\dot{\boldsymbol{R}}_{AB} = [{}_A\boldsymbol{\omega}_{AB}]_\times\, \boldsymbol{R}_{AB} \quad \text{(angular velocity in frame } A) \tag{8}$$

$$\dot{\boldsymbol{R}}_{AB} = \boldsymbol{R}_{AB}\, [{}_B\boldsymbol{\omega}_{AB}]_\times \quad \text{(angular velocity in frame } B) \tag{9}$$

From the equation above, $[{}_A\boldsymbol{\omega}_{AB}]_\times = \dot{\boldsymbol{R}}_{AB}\, \boldsymbol{R}_{AB}^\top$ holds [2]. The angular velocity satisfies the composition property: $\boldsymbol{\omega}_{AC} = \boldsymbol{\omega}_{AB} + \boldsymbol{\omega}_{BC}$ (all expressed in the same frame). **Euler angle rates to angular**

[1] When $\theta = \pm\pi/2$, one DoF is lost — the first and third rotation axes align, making the representation singular. This is analogous to a rank-deficient Jacobian in optimization.

[2] $\dot{\boldsymbol{R}}\,\boldsymbol{R}^\top$ is always skew-symmetric since $\frac{d}{dt}(\boldsymbol{R}\boldsymbol{R}^\top) = \boldsymbol{0}$.

velocity (for ZYX convention) is ${}_I\boldsymbol{\omega}_{IB} = \boldsymbol{E}(\boldsymbol{\Phi})\,\dot{\boldsymbol{\Phi}}$ where $\boldsymbol{E}(\boldsymbol{\Phi}) \in \mathbb{R}^{3\times3}$ maps Euler angle rates to angular velocity. For the ZYX convention with $\boldsymbol{\Phi} = [\phi, \theta, \psi]^\top$ (roll, pitch, yaw), the angular velocity in the inertial frame is a sum of contributions from each Euler angle rate, each about its respective (intermediate) axis:

$$ {}_I\boldsymbol{\omega}_{IB} = \dot{\psi}\,\hat{\boldsymbol{z}}_I + \dot{\theta}\,\boldsymbol{R}_z(\psi)\,\hat{\boldsymbol{y}} + \dot{\phi}\,\boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\,\hat{\boldsymbol{x}} \tag{10} $$

The three terms rotate about the fixed z-axis ($\dot{\psi}$), the y-axis after the z-rotation ($\dot{\theta}$), and the x-axis after both z- and y-rotations ($\dot{\phi}$), respectively. Each unit vector in the inertial frame is:

$$ \hat{\boldsymbol{z}}_I = [0,0,1]^\top, \boldsymbol{R}_z(\psi)\,\hat{\boldsymbol{y}} = [-\sin\psi, \cos\psi, 0]^\top, \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\,\hat{\boldsymbol{x}} = [\cos\psi\cos\theta, \sin\psi\cos\theta, -\sin\theta]^\top \tag{11} $$

These, stacked as columns in the order $\dot{\phi}, \dot{\theta}, \dot{\psi}$, yield the **E matrix**:

$$ \boldsymbol{E}(\boldsymbol{\Phi}) = \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi & 0 \\ \sin\psi\cos\theta & \cos\psi & 0 \\ -\sin\theta & 0 & 1 \end{bmatrix} \tag{12} $$

where $\det(\boldsymbol{E}) = \cos\theta$, so $\boldsymbol{E}$ is singular when $\theta = \pm\pi/2$ (gimbal lock). The useful identities for the cross-product matrix are:

$$ [\boldsymbol{a}]_\times^2 = \boldsymbol{a}\,\boldsymbol{a}^\top - \boldsymbol{a}^\top\boldsymbol{a}\,\boldsymbol{I}_3 = \boldsymbol{a}\,\boldsymbol{a}^\top - \|\boldsymbol{a}\|^2\,\boldsymbol{I}_3, \quad [\boldsymbol{a}]_\times\,\boldsymbol{b} = -[\boldsymbol{b}]_\times\,\boldsymbol{a}, \quad [\boldsymbol{a}]_\times\,[\boldsymbol{b}]_\times = \boldsymbol{b}\,\boldsymbol{a}^\top - (\boldsymbol{a}^\top\boldsymbol{b})\,\boldsymbol{I}_3 \tag{13} $$

2.2.6 Rotation Representations: Summary

Representation	Parameters	Singularity-free?	Compact?
Rotation matrix $\boldsymbol{R}$	9 (6 constraints)	Yes	No
Euler angles $\boldsymbol{\Phi}$	3	No (gimbal lock)	Yes
Angle-axis $\boldsymbol{\phi}$	3	No ($\theta = 0$)	Yes
Unit quaternion $\bar{q}$	4 (1 constraint)	Yes	Yes

- **Unit quaternion:** $\bar{q} = [\eta, \boldsymbol{\epsilon}^\top]^\top \in \mathbb{R}^4$ with $\|\bar{q}\| = 1$, where $\eta = \cos(\theta/2)$ and $\boldsymbol{\epsilon} = \sin(\theta/2)\,\hat{\boldsymbol{n}}$.
- **Quaternion to rotation matrix:** $\boldsymbol{R}(\bar{q}) = (2\eta^2 - 1)\,\boldsymbol{I}_3 + 2\eta\,[\boldsymbol{\epsilon}]_\times + 2\,\boldsymbol{\epsilon}\,\boldsymbol{\epsilon}^\top$
- **Quaternion multiplication** (Hamilton product, corresponds to rotation composition):

$$ \bar{q}_1 \otimes \bar{q}_2 = \begin{bmatrix} \eta_1\eta_2 - \boldsymbol{\epsilon}_1^\top\boldsymbol{\epsilon}_2 \\ \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2 \end{bmatrix} \tag{14} $$

- **Inverse rotation:** $\bar{q}^{-1} = [\eta, -\boldsymbol{\epsilon}^\top]^\top$ (just negate the vector part).
- **Angular velocity from quaternion derivative:**

$$ \dot{\bar{q}} = \frac{1}{2}\begin{bmatrix} -\boldsymbol{\epsilon}^\top \\ \eta\,\boldsymbol{I}_3 + [\boldsymbol{\epsilon}]_\times \end{bmatrix}\boldsymbol{\omega} = \frac{1}{2}\,\boldsymbol{Q}(\bar{q})\,\boldsymbol{\omega} \tag{15} $$

Quaternions are preferred in simulation and control because they avoid gimbal lock and the $\boldsymbol{E}$ matrix singularity, and are widely used in practice, e.g., in simulation frameworks, UAVs, and legged robots.

2.3 Homogeneous Transformations

A 4×4 **homogeneous transformation matrix** combines rotation and translation:

$$ \boldsymbol{T}_{AB} = \begin{bmatrix} \boldsymbol{R}_{AB} & {}_A\boldsymbol{r}_{OA\to OB} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{16} $$

The properties of the transformation matrix are the following:

- **Homogeneous coordinates:** $\bar{\boldsymbol{r}} = [\boldsymbol{r}^\top, 1]^\top$ so that ${}_A\bar{\boldsymbol{r}} = \boldsymbol{T}_{AB}\,{}_B\bar{\boldsymbol{r}}$.
- **Composition:** $\boldsymbol{T}_{AC} = \boldsymbol{T}_{AB}\,\boldsymbol{T}_{BC}$.
- **Inverse:**
$$\boldsymbol{T}_{AB}^{-1} = \boldsymbol{T}_{BA} = \begin{bmatrix} \boldsymbol{R}_{AB}^\top & -\boldsymbol{R}_{AB}^\top\,{}_A\boldsymbol{r} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \tag{17}$$
- **Wrench transformation:** if a wrench $\boldsymbol{w} = [\boldsymbol{f}^\top, \boldsymbol{\tau}^\top]^\top$ is expressed at frame B and we want it at frame A:
$${}_A\boldsymbol{w} = \boldsymbol{X}_{AB}\,{}_B\boldsymbol{w}, \qquad \boldsymbol{X}_{AB} = \begin{bmatrix} \boldsymbol{R}_{AB} & \boldsymbol{0} \\ [{}_A\boldsymbol{r}_{AB}]_\times\,\boldsymbol{R}_{AB} & \boldsymbol{R}_{AB} \end{bmatrix} \tag{18}$$
This is the **force transformation matrix** (adjoint transpose).
- **Twist transformation:** a twist $\boldsymbol{\xi} = [\boldsymbol{v}^\top, \boldsymbol{\omega}^\top]^\top$ transforms as:
$${}_A\boldsymbol{\xi} = \boldsymbol{X}_{AB}^{-\top}\,{}_B\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{R}_{AB} & [{}_A\boldsymbol{r}_{AB}]_\times\,\boldsymbol{R}_{AB} \\ \boldsymbol{0} & \boldsymbol{R}_{AB} \end{bmatrix} {}_B\boldsymbol{\xi} \tag{19}$$

Wrenches transform differently from twists, and the frame invariance of power is a key property: $\boldsymbol{w}^\top\boldsymbol{\xi} = {}_A\boldsymbol{w}^\top\,{}_A\boldsymbol{\xi} = {}_B\boldsymbol{w}^\top\,{}_B\boldsymbol{\xi}$.

2.4 Velocity of a Point on a Moving Body

2.4.1 Transport Equation

For a point P rigidly attached to body B, which moves with angular velocity $\boldsymbol{\omega}_{IB}$ and has reference point velocity $\boldsymbol{v}_B$, the velocity in the inertial frame I is:

$$\boxed{{}_I\boldsymbol{v}_P = {}_I\boldsymbol{v}_B + {}_I\boldsymbol{\omega}_{IB} \times {}_I\boldsymbol{r}_{BP}} \tag{20}$$

This can equivalently be written using the skew-symmetric operator as ${}_I\boldsymbol{v}_P = {}_I\boldsymbol{v}_B + [{}_I\boldsymbol{\omega}_{IB}]_\times\,{}_I\boldsymbol{r}_{BP}$. The **general transport equation** for a time-varying vector $\boldsymbol{r}(t)$ observed from rotating frame B is $\frac{{}^I d}{dt}\boldsymbol{r} = \frac{{}^B d}{dt}\boldsymbol{r} + \boldsymbol{\omega}_{IB} \times \boldsymbol{r}$. This follows from expressing $\boldsymbol{r}(t)$ in rotating frame B as ${}_I\boldsymbol{r} = \boldsymbol{R}_{IB}\,{}_B\boldsymbol{r}$ and differentiating:

$$\frac{{}^I d}{dt}{}_I\boldsymbol{r} = \dot{\boldsymbol{R}}_{IB}\,{}_B\boldsymbol{r} + \boldsymbol{R}_{IB}\,\frac{{}^B d}{dt}{}_B\boldsymbol{r} = [\boldsymbol{\omega}_{IB}]_\times\,\boldsymbol{R}_{IB}\,{}_B\boldsymbol{r} + \boldsymbol{R}_{IB}\,\frac{{}^B d}{dt}{}_B\boldsymbol{r} = \boldsymbol{\omega}_{IB} \times {}_I\boldsymbol{r} + \boldsymbol{R}_{IB}\,\frac{{}^B d}{dt}{}_B\boldsymbol{r}$$

The first term is the rotational contribution; the second is the "body-frame derivative" re-expressed in the inertial frame.

2.4.2 Acceleration of a Point on a Moving Body

The time derivative of Eq. (20) yields:

$$\boxed{{}_I\boldsymbol{a}_P = {}_I\boldsymbol{a}_B + {}_I\boldsymbol{\alpha}_{IB} \times {}_I\boldsymbol{r}_{BP} + {}_I\boldsymbol{\omega}_{IB} \times ({}_I\boldsymbol{\omega}_{IB} \times {}_I\boldsymbol{r}_{BP})} \tag{21}$$

where $\boldsymbol{\alpha}_{IB} = \dot{\boldsymbol{\omega}}_{IB}$ is the angular acceleration. The three terms are:

- **Reference point acceleration:** $\boldsymbol{a}_B$
- **Tangential acceleration:** $\boldsymbol{\alpha} \times \boldsymbol{r}_{BP}$ (due to angular acceleration)
- **Centripetal acceleration:** $\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}_{BP})$

If P also moves relative to B (i.e., $\boldsymbol{r}_{BP}$ is time-varying in frame B), an additional Coriolis term $2\boldsymbol{\omega}_{IB} \times {}_B\dot{\boldsymbol{r}}_{BP}$ appears. The **full acceleration with relative motion** (e.g., point sliding on a rotating body) is calculated as:

$$\boldsymbol{a}_P = \boldsymbol{a}_B + \boldsymbol{\alpha} \times \boldsymbol{r}_{BP} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}_{BP}) + 2\boldsymbol{\omega} \times {}_B\dot{\boldsymbol{r}}_{BP} + {}_B\ddot{\boldsymbol{r}}_{BP} \tag{22}$$

where the five terms are (in order):

- **Reference point acceleration:** $\boldsymbol{a}_B$
- **Tangential acceleration:** due to angular acceleration $\boldsymbol{\alpha}$
- **Centripetal acceleration:** always points toward the rotation axis
- **Coriolis acceleration:** $2\boldsymbol{\omega} \times {}_B\dot{\boldsymbol{r}}_{BP}$, due to relative motion in the rotating frame
- **Relative acceleration:** ${}_B\ddot{\boldsymbol{r}}_{BP}$, acceleration in the body frame

For a point *fixed* on the body (${}_B\dot{\boldsymbol{r}}_{BP} = \mathbf{0}$), terms 4 and 5 vanish, recovering Eq. (21).

2.5 Kinematics of Multi-Body Systems

2.5.1 Joint Types

- **Revolute (R):** rotation about a fixed axis. 1 DoF. Generalized coordinate = joint angle q_i.
- **Prismatic (P):** translation along a fixed axis. 1 DoF. Generalized coordinate = displacement q_i.

2.5.2 Forward Kinematics

For a serial chain with n joints, the end-effector pose relative to the base is:

$$\boldsymbol{T}_{0n}(\boldsymbol{q}) = \boldsymbol{T}_{01}(q_1)\,\boldsymbol{T}_{12}(q_2)\,\cdots\,\boldsymbol{T}_{(n-1)n}(q_n) = \prod_{i=1}^{n} \boldsymbol{T}_{(i-1)i}(q_i) \tag{23}$$

Each $\boldsymbol{T}_{(i-1)i}(q_i)$ depends on one joint variable. For a revolute joint, q_i enters through the rotation part; for a prismatic joint, through the translation part. As an example, for a **2-link planar manipulator** with two revolute joints in the xy-plane and link lengths l_1, l_2:

$$\boldsymbol{T}_{02} = \boldsymbol{T}_{01}(q_1)\boldsymbol{T}_{12}(q_2) = \begin{bmatrix} c_{12} & -s_{12} & 0 & l_1c_1 + l_2c_{12} \\ s_{12} & c_{12} & 0 & l_1s_1 + l_2s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{24}$$

where $c_1 = \cos q_1$, $c_{12} = \cos(q_1 + q_2)$, etc. The end-effector position is:

$$\boldsymbol{r}_{ee} = \begin{bmatrix} l_1 \cos q_1 + l_2 \cos(q_1 + q_2) \\ l_1 \sin q_1 + l_2 \sin(q_1 + q_2) \end{bmatrix} \tag{25}$$

The **Jacobian** (2D, linear velocity only) is:

$$\boldsymbol{J} = \begin{bmatrix} -l_1s_1 - l_2s_{12} & -l_2s_{12} \\ l_1c_1 + l_2c_{12} & l_2c_{12} \end{bmatrix} \tag{26}$$

The determinant $\det(\boldsymbol{J}) = l_1 l_2 \sin q_2$ reveals singularities at $q_2 = 0$ or $q_2 = \pi$, which means the arm is fully extended or folded.

2.5.3 Denavit–Hartenberg (DH) Convention

The **Denavit–Hartenberg (DH) convention** provides a systematic way to assign frames and parameterize each joint transform using 4 parameters per joint: θ_i (joint angle), d_i (link offset), a_i (link length), and α_i (link twist):

$$\boldsymbol{T}_{(i-1)i} = \underbrace{\boldsymbol{R}_z(\theta_i)\,\mathrm{Trans}_z(d_i)}_{\text{joint axis}} \cdot \underbrace{\mathrm{Trans}_x(a_i)\,\boldsymbol{R}_x(\alpha_i)}_{\text{link geometry}} \tag{27}$$

The expanded form is:

$$\boldsymbol{T}_{(i-1)i} = \begin{bmatrix} c\theta_i & -s\theta_i\, c\alpha_i & s\theta_i\, s\alpha_i & a_i\, c\theta_i \\ s\theta_i & c\theta_i\, c\alpha_i & -c\theta_i\, s\alpha_i & a_i\, s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{28}$$

where $c\theta_i = \cos\theta_i$, $s\theta_i = \sin\theta_i$, etc. Variables are $\theta_i = q_i$ for a revolute joint and $d_i = q_i$ for a prismatic joint.

2.5.4 Branched and Closed Chains

The following chain types are common in multi-body systems:

- **Serial (open) chain:** each link has exactly one parent. The end-effector pose is uniquely determined by $\boldsymbol{q}$ via Eq. (23).
- **Branched chain:** a link may have multiple children (e.g., humanoid torso $\rightarrow$ left arm, right arm, head). Each branch uses independent forward kinematics from the branching point.
- **Closed chain:** kinematic loops (e.g., parallel robots, four-bar linkages) introduce constraints, so the generalized coordinates are no longer independent. This requires additional constraint equations $\boldsymbol{\phi}(\boldsymbol{q}) = \mathbf{0}$.

A **spatial twist** $\boldsymbol{\xi} = [\boldsymbol{v}^\top, \boldsymbol{\omega}^\top]^\top \in \mathbb{R}^6$ combines linear and angular velocity, and its dual quantity is the **wrench** (force–torque pair): $\boldsymbol{w} = [\boldsymbol{f}^\top, \boldsymbol{\tau}^\top]^\top \in \mathbb{R}^6$. The power is given by the inner product $P = \boldsymbol{w}^\top \boldsymbol{\xi} = \boldsymbol{f}^\top \boldsymbol{v} + \boldsymbol{\tau}^\top \boldsymbol{\omega}$.

2.6 Jacobian

The **Jacobian** maps joint velocities to end-effector velocities and is the fundamental tool connecting joint space and task space.

2.6.1 Geometric Jacobian

The **geometric Jacobian** $\boldsymbol{J}_G(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$ maps joint velocities to the end-effector twist (linear + angular velocity):

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} = \boldsymbol{J}_G(\boldsymbol{q})\, \dot{\boldsymbol{q}} \tag{29}$$

The i-th column of $\boldsymbol{J}_G$ depends on the joint type:

- **Revolute joint** i (axis $\hat{\boldsymbol{z}}_i$):

$$\boldsymbol{j}_i = \begin{bmatrix} \hat{\boldsymbol{z}}_i \times (\boldsymbol{r}_{ee} - \boldsymbol{r}_i) \\ \hat{\boldsymbol{z}}_i \end{bmatrix} \tag{30}$$

- **Prismatic joint** i (axis $\hat{\boldsymbol{z}}_i$):

$$\boldsymbol{j}_i = \begin{bmatrix} \hat{\boldsymbol{z}}_i \\ \mathbf{0} \end{bmatrix} \tag{31}$$

where $\hat{\boldsymbol{z}}_i$ is the joint axis, $\boldsymbol{r}_{ee}$ is the end-effector position, and $\boldsymbol{r}_i$ is the origin of frame i, all expressed in the same frame.

2.6.2 Analytical Jacobian

The **analytical Jacobian** $\boldsymbol{J}_A(\boldsymbol{q}) \in \mathbb{R}^{m\times n}$ maps joint velocities to the time derivative of a minimal task-space representation $\boldsymbol{x} = [\boldsymbol{r}^\top, \boldsymbol{\Phi}^\top]^\top$ (position + orientation parameters): $\dot{\boldsymbol{x}} = [\dot{\boldsymbol{r}}^\top, \dot{\boldsymbol{\Phi}}^\top]^\top = \boldsymbol{J}_A(\boldsymbol{q})\, \dot{\boldsymbol{q}}$.

2.6.3 Relationship Between Geometric and Analytical Jacobians

Since $\boldsymbol{v} = \dot{\boldsymbol{r}}$ but $\boldsymbol{\omega} \neq \dot{\boldsymbol{\Phi}}$ in general, i.e., angular velocity $\neq$ Euler angle rates, we have $\boldsymbol{\omega} = \boldsymbol{E}(\boldsymbol{\Phi})\, \dot{\boldsymbol{\Phi}}$, and therefore:

$$\boldsymbol{J}_G = \begin{bmatrix} \boldsymbol{I}_3 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{E}(\boldsymbol{\Phi}) \end{bmatrix} \boldsymbol{J}_A, \qquad \boldsymbol{J}_A = \begin{bmatrix} \boldsymbol{I}_3 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{E}(\boldsymbol{\Phi})^{-1} \end{bmatrix} \boldsymbol{J}_G \tag{32}$$

$\boldsymbol{J}_A$ is undefined at Euler angle singularities where $\boldsymbol{E}$ is singular.

2.6.4 Time Derivative of the Jacobian

The end-effector acceleration involves $\dot{\boldsymbol{J}}$: $\ddot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}}$.

2.6.5 Manipulability

The **manipulability ellipsoid** characterizes how well the robot can move in different task-space directions at a given configuration. For unit joint velocities $\|\dot{\boldsymbol{q}}\| = 1$, the task-space velocities satisfy $\dot{\boldsymbol{x}}^\top (\boldsymbol{J}\boldsymbol{J}^\top)^{-1}\dot{\boldsymbol{x}} \leq 1$, which defines an ellipsoid in task space whose semi-axes are the singular values σ_i of $\boldsymbol{J}$ and whose principal directions are the left singular vectors. **Yoshikawa's manipulability measure** is defined as:

$$w(\boldsymbol{q}) = \sqrt{\det(\boldsymbol{J}(\boldsymbol{q})\,\boldsymbol{J}(\boldsymbol{q})^\top)} = \prod_i \sigma_i \tag{33}$$

where $w = 0$ at singularities; maximizing w provides isotropic motion capability. The **condition number** $\kappa(\boldsymbol{J}) = \sigma_{\max}/\sigma_{\min}$ is another measure: $\kappa = 1$ means isotropic (equally capable in all directions), while $\kappa \to \infty$ indicates proximity to a singularity. It is useful for workspace analysis and trajectory optimization.

2.6.6 Singularity Types

The following types of singularities arise in practice:

- **Boundary singularities:** Occur at the workspace boundary (arm fully extended or fully retracted). Some directions become unreachable.
- **Interior singularities:** Occur inside the workspace (e.g., when two joint axes align). Motion capability is lost in specific directions.
- **Architectural singularities:** Due to the robot's kinematic structure (e.g., when rotation axes intersect at a common point — wrist singularity).

As an example, for the **2-link planar** manipulator, $\det(\boldsymbol{J}) = l_1 l_2 \sin q_2 = 0$ when $q_2 = 0$, i.e., fully extended, or $q_2 = \pi$, i.e., fully folded. At $q_2 = 0$, the robot cannot move along the arm direction, which is a boundary singularity.

2.6.7 Jacobian in Different Frames

The geometric Jacobian depends on the frame of expression: if $\boldsymbol{J}_0$ is expressed in the base frame, its expression in frame k is $\boldsymbol{J}_k = \mathrm{diag}(\boldsymbol{R}_{k0}, \boldsymbol{R}_{k0})\,\boldsymbol{J}_0$. The end-effector-frame Jacobian (body Jacobian) is often used for body-frame control.

2.6.8 Static Force Relationship

At static equilibrium ($\ddot{\boldsymbol{q}} = \dot{\boldsymbol{q}} = \boldsymbol{0}$), the Jacobian transpose relates end-effector forces to joint torques: $\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f}_{\text{ext}}$. This duality with the velocity relationship $\boldsymbol{v} = \boldsymbol{J}\,\dot{\boldsymbol{q}}$ follows from the principle of virtual work: $\delta W = \boldsymbol{f}^\top \delta\boldsymbol{r} = \boldsymbol{f}^\top \boldsymbol{J}\,\delta\boldsymbol{q} = \boldsymbol{\tau}^\top \delta\boldsymbol{q}$.

2.6.9 Jacobian Transpose Control

Jacobian transpose control is a simple task-space controller that avoids computing the Jacobian inverse:

$$\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f}_d + \boldsymbol{g}(\boldsymbol{q}), \qquad \boldsymbol{f}_d = \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x}) - \boldsymbol{K}_d\,\dot{\boldsymbol{x}} \tag{34}$$

This applies a virtual spring-damper force at the end-effector, projected to joint torques via $\boldsymbol{J}^\top$, and is always well-defined (no singularity issues), though convergence is slower because the effective gain depends on $\boldsymbol{J}$.

2.7 Kinematic Control

2.7.1 Inverse Kinematics (IK)

Given a desired end-effector velocity $\dot{\boldsymbol{x}}_d$, the joint velocities are $\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}\,\dot{\boldsymbol{x}}_d$, which requires $\boldsymbol{J}$ to be square and non-singular.

2.7.2 Redundant Systems ($n > m$)

When the robot has more joints than task-space dimensions, the **right pseudoinverse** (minimum-norm solution) applies:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}\,\dot{\boldsymbol{x}}_d, \qquad \boldsymbol{J}^{+} = \boldsymbol{J}^{\top}(\boldsymbol{J}\,\boldsymbol{J}^{\top})^{-1} \tag{35}$$

2.7.3 Overdetermined Systems ($n < m$)

The **left pseudoinverse** (least-squares solution) gives $\dot{\boldsymbol{q}} = (\boldsymbol{J}^{\top}\boldsymbol{J})^{-1}\boldsymbol{J}^{\top}\,\dot{\boldsymbol{x}}_d$.

2.7.4 Singularities

A singularity occurs when $\boldsymbol{J}$ loses rank, i.e., $\det(\boldsymbol{J}\,\boldsymbol{J}^{\top}) \to 0$. At singularities:

- Certain task-space directions become unachievable
- Joint velocities $\to \infty$ to achieve finite task-space velocities
- The pseudoinverse becomes numerically unstable

The **damped least squares** (Levenberg–Marquardt) regularization addresses this:

$$\boxed{\dot{\boldsymbol{q}} = \boldsymbol{J}^{\top}(\boldsymbol{J}\,\boldsymbol{J}^{\top} + \lambda^2\boldsymbol{I})^{-1}\,\dot{\boldsymbol{x}}_d} \tag{36}$$

This limits joint velocities near singularities at the cost of tracking accuracy, where the damping $\lambda > 0$ trades off accuracy vs. feasibility.

2.7.5 Redundancy Resolution

For redundant robots ($n > m$), the null space $(\boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J})$ allows secondary objectives without affecting the primary task:

$$\boxed{\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}\,\dot{\boldsymbol{x}}_d + (\boldsymbol{I}_n - \boldsymbol{J}^{+}\boldsymbol{J})\,\dot{\boldsymbol{q}}_0} \tag{37}$$

where $\dot{\boldsymbol{q}}_0 \in \mathbb{R}^n$ is an arbitrary joint velocity that is projected onto the null space of $\boldsymbol{J}$. Typical choices for $\dot{\boldsymbol{q}}_0$ include:

- Gradient of a cost function, e.g., joint limit avoidance, manipulability maximization,
- Default posture: $\dot{\boldsymbol{q}}_0 = k(\boldsymbol{q}_{\text{default}} - \boldsymbol{q})$

2.7.6 Position-Level Inverse Kinematics

The velocity-level methods above solve the *differential* IK problem. For *position-level* IK, i.e., finding $\boldsymbol{q}$ such that $\boldsymbol{x}(\boldsymbol{q}) = \boldsymbol{x}_d$, the Newton–Raphson iteration is:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \boldsymbol{J}(\boldsymbol{q}_k)^{-1}\,(\boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{q}_k)) \tag{38}$$

or with the pseudoinverse for redundant/singular cases:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \boldsymbol{J}^{+}(\boldsymbol{q}_k)\,(\boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{q}_k)) + (\boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J})\,\dot{\boldsymbol{q}}_0 \tag{39}$$

The Newton–Raphson method converges quadratically near the solution when $\boldsymbol{J}$ is non-singular, but may diverge from poor initial guesses or near singularities.

2.7.7 Closed-Loop Inverse Kinematics (CLIK)

Closed-loop inverse kinematics (CLIK) combines feedback with feedforward for robust trajectory tracking:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}(\dot{\boldsymbol{x}}_d + \boldsymbol{K}\,(\boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{q}))) \tag{40}$$

where $\boldsymbol{K}$ is a positive definite gain matrix. The feedback term $\boldsymbol{K}(\boldsymbol{x}_d - \boldsymbol{x})$ ensures convergence even with numerical drift, combining the velocity-level feedforward $\dot{\boldsymbol{x}}_d$ with position-level error correction analogous to a PI controller.

2.7.8 Task-Priority (Hierarchical) Control

When multiple tasks must be satisfied simultaneously with different priorities, a **task-priority framework** is used. For a **two-task hierarchy**, let $\boldsymbol{J}_1$ be the primary task Jacobian and $\boldsymbol{J}_2$ the secondary:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_1^+ \dot{\boldsymbol{x}}_1 + (\boldsymbol{I} - \boldsymbol{J}_1^+ \boldsymbol{J}_1)\, \boldsymbol{J}_2^+ \left(\dot{\boldsymbol{x}}_2 - \boldsymbol{J}_2\, \boldsymbol{J}_1^+ \dot{\boldsymbol{x}}_1\right) \tag{41}$$

The first term achieves the primary task exactly if feasible, while the second term projects the secondary task into the null space of the primary task, ensuring the primary task is never disturbed. The **general k-task hierarchy** is:

$$\dot{\boldsymbol{q}} = \sum_{i=1}^{k} \left(\prod_{j=1}^{i-1} \boldsymbol{N}_j\right) \boldsymbol{J}_i^+ \left(\dot{\boldsymbol{x}}_i - \boldsymbol{J}_i \sum_{l=1}^{i-1} \left(\prod_{j=1}^{l-1} \boldsymbol{N}_j\right) \boldsymbol{J}_l^+ \dot{\boldsymbol{x}}_l\right) \tag{42}$$

where $\boldsymbol{N}_j = \boldsymbol{I} - \boldsymbol{J}_j^+ \boldsymbol{J}_j$ is the null-space projector of task j. This is widely used in humanoid and legged robot control, e.g., task 1: maintain balance, task 2: track hand position, task 3: preferred posture.

2.7.9 Weighted Pseudoinverse

Large velocities in certain joints, e.g., near joint limits, can be penalized using the **weighted pseudoinverse**:

$$\boldsymbol{J}_{\boldsymbol{W}}^+ = \boldsymbol{W}^{-1} \boldsymbol{J}^\top (\boldsymbol{J}\, \boldsymbol{W}^{-1} \boldsymbol{J}^\top)^{-1} \tag{43}$$

where $\boldsymbol{W} \succ \boldsymbol{0}$ is a positive definite weight matrix. This minimizes $\dot{\boldsymbol{q}}^\top \boldsymbol{W} \dot{\boldsymbol{q}}$ subject to $\boldsymbol{J}\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}_d$. A large W_{ii} penalizes large $\dot{q}_i$, so W_{ii} is increased near joint limits.

2.7.10 Summary of IK Methods

Method	When to use	Pros	Cons
$\boldsymbol{J}^{-1}$	Square, non-singular	Exact, fast	Fails at singularities
$\boldsymbol{J}^+$ (right)	Redundant ($n > m$)	Min. norm solution	Unstable near sing.
$\boldsymbol{J}^+$ (left)	Overconstrained ($n < m$)	Least-squares	Tracking error
Damped LS	Any, near singularities	Robust	Tracking error
Null space	Redundant, multi-task	Secondary objectives	More complex
Task priority	Multiple tasks	Strict hierarchy	Computational cost

2.8 Floating Base Kinematics

For robots with an unactuated base, e.g., legged robots, drones, the base is free to move in space. The **generalized coordinates** are:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{r}_{IB} \\ \boldsymbol{\Phi} \\ \boldsymbol{q}_j \end{bmatrix} \in \mathbb{R}^n \tag{44}$$

where $n = 6 + n_j$, $\boldsymbol{r}_{IB} \in \mathbb{R}^3$ is the base position in the inertial frame, $\boldsymbol{\Phi} \in \mathbb{R}^3$ is the base orientation, e.g., Euler angles, and $\boldsymbol{q}_j \in \mathbb{R}^{n_j}$ is the joint angles. The **generalized velocities** are:

$$\boldsymbol{u} = \begin{bmatrix} {}_I\boldsymbol{v}_B \\ {}_I\boldsymbol{\omega}_{IB} \\ \dot{\boldsymbol{q}}_j \end{bmatrix} \in \mathbb{R}^n \tag{45}$$

Here $\boldsymbol{u} \neq \dot{\boldsymbol{q}}$ because $\boldsymbol{\omega} \neq \dot{\boldsymbol{\Phi}}$ [3]. The **contact Jacobian** for a foot (or contact point) c on the robot is $\boldsymbol{v}_c = \boldsymbol{J}_c(\boldsymbol{q})\,\boldsymbol{u}$. In stance, $\boldsymbol{v}_c = \boldsymbol{0}$, giving a kinematic constraint $\boldsymbol{J}_c\,\boldsymbol{u} = \boldsymbol{0}$.

The **support-consistent Jacobian** for a floating-base robot with n_c contact points in stance maps joint velocities to the velocity of any point on the robot *given that the contacts are fixed*. The contact constraint $\boldsymbol{J}_c\boldsymbol{u} = \boldsymbol{0}$ constrains the motion as $\boldsymbol{u} = (\boldsymbol{I} - \boldsymbol{J}_c^{+}\boldsymbol{J}_c)\,\boldsymbol{u}_{\text{free}}$, where $\boldsymbol{u}_{\text{free}}$ is the unconstrained velocity and the projection $(\boldsymbol{I} - \boldsymbol{J}_c^{+}\boldsymbol{J}_c)$ removes components that violate contact. The robot's center of mass (projected onto the ground plane) must lie within the **support polygon**, i.e., the convex hull of the contact points, for static stability, which is a necessary condition for quasi-static balance. For dynamic balance, the ZMP must lie within the support polygon. In locomotion, legs alternate between **stance**, i.e., foot fixed on ground, and **swing**, i.e., foot moving through air. The kinematic structure changes at each contact transition, making locomotion a *hybrid dynamical system*.

3 Dynamics

3.1 Foundations of Classical Mechanics

3.1.1 Virtual Displacements and Virtual Work

A **virtual displacement** $\delta\boldsymbol{r}_i$ is an infinitesimal displacement consistent with the system's constraints, occurring at a fixed instant in time. In a system with generalized coordinates $\boldsymbol{q}$, the virtual displacement is:

$$\delta\boldsymbol{r}_i = \sum_{j=1}^{n} \frac{\partial \boldsymbol{r}_i}{\partial q_j}\,\delta q_j = \boldsymbol{J}_i\,\delta\boldsymbol{q} \tag{46}$$

The **virtual work** done by forces $\boldsymbol{f}_i$ through virtual displacements is:

$$\delta W = \sum_i \boldsymbol{f}_i^{\top}\,\delta\boldsymbol{r}_i = \sum_i \boldsymbol{f}_i^{\top}\,\boldsymbol{J}_i\,\delta\boldsymbol{q} = \boldsymbol{Q}^{\top}\,\delta\boldsymbol{q} \tag{47}$$

where $\boldsymbol{Q} \in \mathbb{R}^n$ are the **generalized forces** $Q_j = \sum_i \boldsymbol{f}_i^{\top}\,\frac{\partial \boldsymbol{r}_i}{\partial q_j}$.

3.1.2 d'Alembert's Principle

Treating inertial terms as additional applied forces, the following holds for all virtual displacements $\delta\boldsymbol{q}$ consistent with constraints:

$$\boxed{\sum_i (\boldsymbol{f}_i - m_i\,\ddot{\boldsymbol{r}}_i)^{\top}\,\delta\boldsymbol{r}_i = 0} \tag{48}$$

Since the δq_j are independent, each coefficient must vanish:

$$\sum_i \boldsymbol{f}_i^{\top}\,\frac{\partial \boldsymbol{r}_i}{\partial q_j} = \sum_i m_i\,\ddot{\boldsymbol{r}}_i^{\top}\,\frac{\partial \boldsymbol{r}_i}{\partial q_j}, \quad j = 1, \ldots, n \tag{49}$$

3.1.3 From d'Alembert to Lagrange

From d'Alembert's principle (Eq. (48)) and $\boldsymbol{r}_i = \boldsymbol{r}_i(\boldsymbol{q})$, the following holds:

$$\dot{\boldsymbol{r}}_i = \sum_j \frac{\partial \boldsymbol{r}_i}{\partial q_j}\,\dot{q}_j, \qquad \frac{\partial \dot{\boldsymbol{r}}_i}{\partial \dot{q}_j} = \frac{\partial \boldsymbol{r}_i}{\partial q_j} \tag{50}$$

Using the identity $\frac{d}{dt}\frac{\partial \boldsymbol{r}_i}{\partial q_j} = \frac{\partial \dot{\boldsymbol{r}}_i}{\partial q_j}$ and the kinetic energy $T = \sum_i \frac{1}{2} m_i\,\dot{\boldsymbol{r}}_i^{\top}\dot{\boldsymbol{r}}_i$, one can show:

$$\sum_i m_i\,\ddot{\boldsymbol{r}}_i^{\top}\,\frac{\partial \boldsymbol{r}_i}{\partial q_j} = \frac{d}{dt}\frac{\partial T}{\partial \dot{q}_j} - \frac{\partial T}{\partial q_j} \tag{51}$$

[3] The angular velocity is not the time derivative of Euler angles.

If forces are derived from a potential U, i.e., $\boldsymbol{f}_i = -\nabla_{\boldsymbol{r}_i} U$, then $Q_j = -\partial U / \partial q_j$ holds, and since U does not depend on $\dot{\boldsymbol{q}}$, the following holds:

$$\frac{d}{dt}\frac{\partial (T-U)}{\partial \dot{q}_j} - \frac{\partial (T-U)}{\partial q_j} = \tau_j \tag{52}$$

This is the Euler–Lagrange equation, i.e., Eq. (66), with $L = T - U$.

3.1.4 Generalized Forces for Common Force Types

- **Gravity:** $Q_j^{\text{grav}} = -\partial U / \partial q_j$ where $U = -\sum_i m_i \boldsymbol{g}^\top \boldsymbol{r}_{c_i}$
- **Joint torque τ_j:** acts directly on q_j, so $Q_j = \tau_j$
- **External force $\boldsymbol{f}^{\text{ext}}$ at point P:** $Q_j = \boldsymbol{f}^{\text{ext}\top} \frac{\partial \boldsymbol{r}_P}{\partial q_j} = \boldsymbol{f}^{\text{ext}\top} \boldsymbol{J}_{P,j}$ (column j of the Jacobian at point P)
- **Spring:** $Q_j = -k\,(q_j - q_{j,0})$ for a spring at joint j
- **Damping:** $Q_j = -d\,\dot{q}_j$ for viscous damping at joint j

3.2 Newton–Euler Method

3.2.1 Newton's Equation (Translation)

For a rigid body with mass m and center-of-mass acceleration $\boldsymbol{a}_c$, the following holds:

$$\boxed{\boldsymbol{f} = m\,\boldsymbol{a}_c} \tag{53}$$

where $\boldsymbol{f}$ is the net external force.

3.2.2 Euler's Equation (Rotation)

For a rigid body rotating with angular velocity $\boldsymbol{\omega}$ and angular acceleration $\boldsymbol{\alpha} = \dot{\boldsymbol{\omega}}$, the torque about the center of mass in the *body frame* is:

$$\boxed{{}_B\boldsymbol{\tau}_c = {}_B\boldsymbol{\Theta}_c\,{}_B\boldsymbol{\alpha} + {}_B\boldsymbol{\omega} \times ({}_B\boldsymbol{\Theta}_c\,{}_B\boldsymbol{\omega})} \tag{54}$$

In skew-symmetric form, this becomes $\boldsymbol{\tau}_c = \boldsymbol{\Theta}_c\,\boldsymbol{\alpha} + [\boldsymbol{\omega}]_\times\,\boldsymbol{\Theta}_c\,\boldsymbol{\omega}$. The Euler equation is expressed in the body frame because the inertia tensor $\boldsymbol{\Theta}_c$ is constant in the body frame.

3.2.3 Inertia Tensor

The **inertia tensor** about the center of mass is:

$$\boldsymbol{\Theta}_c = \int_V \rho(\boldsymbol{r}) \left(\boldsymbol{r}^\top \boldsymbol{r}\, \boldsymbol{I}_3 - \boldsymbol{r}\,\boldsymbol{r}^\top \right) dV = -\int_V \rho(\boldsymbol{r})\, [\boldsymbol{r}]_\times^2\, dV \tag{55}$$

The inertia tensor $\boldsymbol{\Theta}_c$ is symmetric and positive semi-definite, and positive definite for non-degenerate bodies. The **parallel axis theorem** (Steiner's theorem) relates the inertia about a different point:

$$\boldsymbol{\Theta}_O = \boldsymbol{\Theta}_c + m \left(\boldsymbol{r}_{OC}^\top \boldsymbol{r}_{OC}\, \boldsymbol{I}_3 - \boldsymbol{r}_{OC}\, \boldsymbol{r}_{OC}^\top \right) = \boldsymbol{\Theta}_c - m\, [\boldsymbol{r}_{OC}]_\times^2 \tag{56}$$

where $\boldsymbol{r}_{OC}$ is the vector from the new reference point O to the center of mass C. The **change of reference frame** for the inertia tensor is ${}_A\boldsymbol{\Theta} = \boldsymbol{R}_{AB}\,{}_B\boldsymbol{\Theta}\,\boldsymbol{R}_{AB}^\top$. In the body frame, ${}_B\boldsymbol{\Theta}$ is constant, i.e., the body shape doesn't change, while in the inertial frame, ${}_I\boldsymbol{\Theta}(t)$ varies with orientation.

3.2.4 Angular Momentum

The **angular momentum** about the center of mass is $\boldsymbol{h}_c = \boldsymbol{\Theta}_c \boldsymbol{\omega}$. Newton's and Euler's laws can be written compactly as:

$$\boldsymbol{f} = \frac{d}{dt}(m\,\boldsymbol{v}_c) = m\,\boldsymbol{a}_c \qquad \text{(linear momentum)} \tag{57}$$

$$\boldsymbol{\tau}_c = \frac{d}{dt}\boldsymbol{h}_c = \frac{d}{dt}(\boldsymbol{\Theta}_c\,\boldsymbol{\omega}) \qquad \text{(angular momentum)} \tag{58}$$

In the body frame where $\boldsymbol{\Theta}_c$ is constant, ${}_B\boldsymbol{\tau}_c = \boldsymbol{\Theta}_c\,\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{\Theta}_c\,\boldsymbol{\omega}$, recovering Euler's equation Eq. (54). The $\boldsymbol{\omega} \times \boldsymbol{\Theta}_c\,\boldsymbol{\omega}$ term arises because the body frame rotates, i.e., it is the "fictitious" torque from working in a non-inertial frame.

3.2.5 Recursive Newton–Euler Algorithm

The **recursive Newton–Euler algorithm** is an $O(n)$ algorithm for inverse dynamics that performs two passes through the kinematic chain. The **forward pass** (base $\to$ end-effector, $i = 1, \ldots, n$) computes velocities and accelerations of each link:

$$\boldsymbol{\omega}_i = \boldsymbol{R}_{i,i\text{-}1}\,\boldsymbol{\omega}_{i\text{-}1} + \dot{q}_i\,\hat{\boldsymbol{z}}_i \qquad \text{(revolute)} \tag{59}$$

$$\dot{\boldsymbol{\omega}}_i = \boldsymbol{R}_{i,i\text{-}1}\,\dot{\boldsymbol{\omega}}_{i\text{-}1} + \ddot{q}_i\,\hat{\boldsymbol{z}}_i + \boldsymbol{\omega}_i \times \dot{q}_i\,\hat{\boldsymbol{z}}_i \tag{60}$$

$$\boldsymbol{a}_i = \boldsymbol{R}_{i,i\text{-}1}\big(\boldsymbol{a}_{i\text{-}1} + \dot{\boldsymbol{\omega}}_{i\text{-}1} \times \boldsymbol{r}_{(i\text{-}1),i} + \boldsymbol{\omega}_{i\text{-}1} \times (\boldsymbol{\omega}_{i\text{-}1} \times \boldsymbol{r}_{(i\text{-}1),i})\big) \tag{61}$$

$$\boldsymbol{a}_{c_i} = \boldsymbol{a}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i,c_i} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i,c_i}) \tag{62}$$

All quantities are expressed in link i's frame, initialized with $\boldsymbol{\omega}_0 = \boldsymbol{0}$, $\dot{\boldsymbol{\omega}}_0 = \boldsymbol{0}$, and $\boldsymbol{a}_0 = -\boldsymbol{g}$, i.e., gravity as base "acceleration". Setting $\boldsymbol{a}_0 = -\boldsymbol{g}$ instead of adding gravity explicitly to each body works because subtracting gravity from the base acceleration is equivalent to adding gravitational force $m_i\boldsymbol{g}$ to each body by d'Alembert's principle. The **backward pass** (end-effector $\to$ base, $i = n, \ldots, 1$) computes forces and torques:

$$\boldsymbol{f}_i = m_i\,\boldsymbol{a}_{c_i} + \boldsymbol{R}_{i,i+1}\,\boldsymbol{f}_{i+1} \tag{63}$$

$$\boldsymbol{\tau}_i = \boldsymbol{\Theta}_{c_i}\,\dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times \boldsymbol{\Theta}_{c_i}\,\boldsymbol{\omega}_i + \boldsymbol{r}_{i,c_i} \times m_i\,\boldsymbol{a}_{c_i} + \boldsymbol{R}_{i,i+1}\,(\boldsymbol{\tau}_{i+1} + \boldsymbol{r}_{i,i+1} \times \boldsymbol{f}_{i+1}) \tag{64}$$

The joint torque is then $\tau_i = \boldsymbol{\tau}_i^\top\,\hat{\boldsymbol{z}}_i$ (revolute), $f_i = \boldsymbol{f}_i^\top\,\hat{\boldsymbol{z}}_i$ (prismatic). The total complexity is $O(n)$, making this the most efficient algorithm for inverse dynamics in real-time control. The recursive NE algorithm can be used for:

- **Inverse dynamics:** Given $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$, the algorithm directly yields $\boldsymbol{\tau}$.
- **Gravity compensation:** Setting $\dot{\boldsymbol{q}} = \ddot{\boldsymbol{q}} = \boldsymbol{0}$ yields $\boldsymbol{g}(\boldsymbol{q})$ directly.
- **Coriolis/centrifugal:** With $\ddot{\boldsymbol{q}} = \boldsymbol{0}$ and $\boldsymbol{a}_0 = \boldsymbol{0}$ (no gravity), the result is $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, the Coriolis and centrifugal forces (see Section 3.5).
- **Mass matrix column j:** With $\dot{\boldsymbol{q}} = \boldsymbol{0}$, $\boldsymbol{a}_0 = \boldsymbol{0}$, $\ddot{\boldsymbol{q}} = \hat{\boldsymbol{e}}_j$ (the j-th standard basis vector, i.e., all zeros except 1 in position j), the result is column j of $\boldsymbol{M}(\boldsymbol{q})$. Repeating n times gives the full $\boldsymbol{M}$ in $O(n^2)$.

3.3 Lagrange Method

The **Lagrange method** is an energy-based approach that avoids computing internal constraint forces.

3.3.1 Kinetic and Potential Energy

The **kinetic energy** of a rigid body i is $T_i = \frac{1}{2}\,m_i\,\boldsymbol{v}_{c_i}^\top \boldsymbol{v}_{c_i} + \frac{1}{2}\,\boldsymbol{\omega}_i^\top\,\boldsymbol{\Theta}_{c_i}\,\boldsymbol{\omega}_i$, and its total, summing over all n_b bodies, where n_b is the number of rigid bodies in the system, is $T = \sum_{i=1}^{n_b} T_i = \frac{1}{2}\,\dot{\boldsymbol{q}}^\top\,\boldsymbol{M}(\boldsymbol{q})\,\dot{\boldsymbol{q}}$ where the **mass matrix** (generalized inertia matrix) is:

$$\boxed{\boldsymbol{M}(\boldsymbol{q}) = \sum_{i=1}^{n_b} \Big[m_i\,\boldsymbol{J}_{v_i}^\top\,\boldsymbol{J}_{v_i} + \boldsymbol{J}_{\omega_i}^\top\,\boldsymbol{\Theta}_{c_i}\,\boldsymbol{J}_{\omega_i} \Big]} \tag{65}$$

Here $\boldsymbol{J}_{v_i}$ and $\boldsymbol{J}_{\omega_i}$ are the translational and rotational parts of the Jacobian for body i's center of mass: $\boldsymbol{v}_{c_i} = \boldsymbol{J}_{v_i}\,\dot{\boldsymbol{q}}, \quad \boldsymbol{\omega}_i = \boldsymbol{J}_{\omega_i}\,\dot{\boldsymbol{q}}$. The **potential energy** (gravitational) is $U = -\sum_{i=1}^{n_b} m_i\,\boldsymbol{g}^\top\,\boldsymbol{r}_{c_i}(\boldsymbol{q})$ where $\boldsymbol{g}$ is the gravitational acceleration vector, e.g., $[0, 0, -g]^\top$.

3.3.2 Euler–Lagrange Equations

The **Lagrangian** is defined as $L = T - U$, and the equations of motion are:

$$\boxed{\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} - \frac{\partial L}{\partial q_j} = \tau_j, \quad j = 1, \ldots, n} \tag{66}$$

3.3.3 Deriving the Standard EOM from the Lagrangian

From $T = \frac{1}{2}\dot{\boldsymbol{q}}^\top \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}}$ and $U = U(\boldsymbol{q})$, with $L = T - U$:

- **Step 1:** Compute $\frac{\partial L}{\partial \dot{q}_j}$:

$$\frac{\partial L}{\partial \dot{q}_j} = \frac{\partial T}{\partial \dot{q}_j} = \sum_k M_{jk}\,\dot{q}_k = [\boldsymbol{M}\,\dot{\boldsymbol{q}}]_j \tag{67}$$

- **Step 2:** Compute $\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j}$:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} = \sum_k M_{jk}\,\ddot{q}_k + \sum_k \dot{M}_{jk}\,\dot{q}_k = [\boldsymbol{M}\,\ddot{\boldsymbol{q}}]_j + [\dot{\boldsymbol{M}}\,\dot{\boldsymbol{q}}]_j \tag{68}$$

- **Step 3:** Compute $\frac{\partial L}{\partial q_j}$:

$$\frac{\partial L}{\partial q_j} = \frac{1}{2}\sum_{k,l}\frac{\partial M_{kl}}{\partial q_j}\,\dot{q}_k\,\dot{q}_l - \frac{\partial U}{\partial q_j} \tag{69}$$

- **Step 4:** Combine:

$$\tau_j = \sum_k M_{jk}\,\ddot{q}_k + \sum_k \dot{M}_{jk}\,\dot{q}_k - \frac{1}{2}\sum_{k,l}\frac{\partial M_{kl}}{\partial q_j}\,\dot{q}_k\,\dot{q}_l + \frac{\partial U}{\partial q_j} \tag{70}$$

With $\dot{M}_{jk} = \sum_l \frac{\partial M_{jk}}{\partial q_l}\dot{q}_l$ and rearranging the velocity-dependent terms yields the Coriolis matrix with Christoffel symbols Eq. (80), giving $\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}$ which is the standard form of Eq. (79) with $\boldsymbol{b} = \boldsymbol{C}\,\dot{\boldsymbol{q}}$ and $\boldsymbol{g} = \partial U/\partial \boldsymbol{q}$.

3.3.4 Example: 2-Link Planar Manipulator

As an example, for a **2-link planar robot** with masses m_1, m_2, lengths l_1, l_2, center-of-mass distances l_{c_1}, l_{c_2}, and inertias I_1, I_2 about each link's CoM: The **mass matrix** is:

$$\boldsymbol{M} = \begin{bmatrix} M_{11} & M_{12} \\ M_{12} & M_{22} \end{bmatrix} \tag{71}$$

where:

$$M_{11} = m_1 l_{c_1}^2 + m_2(l_1^2 + l_{c_2}^2 + 2 l_1 l_{c_2}\cos q_2) + I_1 + I_2 \tag{72}$$

$$M_{12} = m_2(l_{c_2}^2 + l_1 l_{c_2}\cos q_2) + I_2 \tag{73}$$

$$M_{22} = m_2 l_{c_2}^2 + I_2 \tag{74}$$

The **Coriolis/centrifugal** term is:

$$\boldsymbol{b} = \begin{bmatrix} -m_2 l_1 l_{c_2}\sin q_2\,(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2) \\ m_2 l_1 l_{c_2}\sin q_2\,\dot{q}_1^2 \end{bmatrix} \tag{75}$$

The **gravity** term is:

$$\boldsymbol{g} = \begin{bmatrix} (m_1 l_{c_1} + m_2 l_1) g \cos q_1 + m_2 l_{c_2} g \cos(q_1 + q_2) \\ m_2 l_{c_2} g \cos(q_1 + q_2) \end{bmatrix} \tag{76}$$

This example illustrates how $\boldsymbol{M}$ depends on q_2, i.e., coupling between joints, and how Coriolis terms involve products of velocities.

3.3.5 Comparison of the Three Methods

	Newton–Euler	**Lagrange**	**Projected NE**
Formulation	Force/torque balance	Energy-based	Force + projection
Complexity	$O(n)$ (recursive)	$O(n^3)$ or higher	$O(n^2)$
Gives $\boldsymbol{M}$ explicitly?	No (gives $\boldsymbol{\tau}$)	Yes	Yes
Internal forces?	Computes them	Avoids them	Avoids them
Best for	Real-time inv. dyn.	Symbolic analysis	Deriving EOM

3.4 Projected Newton–Euler Method

The **projected Newton–Euler method** combines the physical intuition of Newton–Euler with the efficiency of working in generalized coordinates. The key idea is to write Newton–Euler equations for each body, then project them into joint space using the Jacobians. By the principle of virtual work, the generalized forces equal:

$$\boxed{\boldsymbol{\tau} = \sum_{i=1}^{n_b} \left[\boldsymbol{J}_{v_i}^\top \left(m_i \, \boldsymbol{a}_{c_i} - \boldsymbol{f}_i^{\text{ext}} \right) + \boldsymbol{J}_{\omega_i}^\top \left(\boldsymbol{\Theta}_{c_i} \, \boldsymbol{\alpha}_i + \boldsymbol{\omega}_i \times \boldsymbol{\Theta}_{c_i} \, \boldsymbol{\omega}_i - \boldsymbol{\tau}_i^{\text{ext}} \right) \right]} \tag{77}$$

where $\boldsymbol{a}_{c_i}$, $\boldsymbol{\omega}_i$, $\boldsymbol{\alpha}_i$ are computed from forward kinematics, and $\boldsymbol{f}_i^{\text{ext}}$, $\boldsymbol{\tau}_i^{\text{ext}}$ are external forces/torques on body i including gravity. Using $\boldsymbol{a}_{c_i} = \boldsymbol{J}_{v_i} \, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{v_i} \, \dot{\boldsymbol{q}}$ and $\boldsymbol{\alpha}_i = \boldsymbol{J}_{\omega_i} \, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{\omega_i} \, \dot{\boldsymbol{q}}$, this directly yields the mass matrix Eq. (65) and the bias/gravity terms. Expanding the projected equation by substituting these into Eq. (77) gives:

$$\begin{aligned} \boldsymbol{\tau} = & \underbrace{\sum_i \left(m_i \boldsymbol{J}_{v_i}^\top \boldsymbol{J}_{v_i} + \boldsymbol{J}_{\omega_i}^\top \boldsymbol{\Theta}_{c_i} \boldsymbol{J}_{\omega_i} \right)}_{=\boldsymbol{M}(\boldsymbol{q})} \ddot{\boldsymbol{q}} + \underbrace{\sum_i \left(m_i \boldsymbol{J}_{v_i}^\top \dot{\boldsymbol{J}}_{v_i} + \boldsymbol{J}_{\omega_i}^\top \boldsymbol{\Theta}_{c_i} \dot{\boldsymbol{J}}_{\omega_i} \right) \dot{\boldsymbol{q}} + \sum_i \boldsymbol{J}_{\omega_i}^\top (\boldsymbol{\omega}_i \times \boldsymbol{\Theta}_{c_i} \boldsymbol{\omega}_i)}_{=\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}})} \\ & \underbrace{- \sum_i \left(\boldsymbol{J}_{v_i}^\top \boldsymbol{f}_i^{\text{ext}} + \boldsymbol{J}_{\omega_i}^\top \boldsymbol{\tau}_i^{\text{ext}} \right)}_{=\boldsymbol{\tau}^{\text{ext}}} \end{aligned} \tag{78}$$

The first term gives the mass matrix $\boldsymbol{M}(\boldsymbol{q})$, the second and third terms give the Coriolis/centrifugal term $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, and the last term gives the external forces projected into joint space including gravity. The advantages of this method include:

- No need to compute partial derivatives of energy, unlike Lagrange
- Works directly with physical forces and torques, like Newton–Euler
- Produces joint-space equations directly, unlike recursive Newton–Euler which gives per-link forces
- Gives $\boldsymbol{M}$, $\boldsymbol{b}$, $\boldsymbol{g}$ explicitly, which is useful for control law implementation

3.5 Equations of Motion

All three methods (Lagrange, Newton–Euler, Projected NE) yield the same **standard form** of the equations of motion:

$$\boxed{\boldsymbol{M}(\boldsymbol{q}) \, \ddot{\boldsymbol{q}} + \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{S}^\top \, \boldsymbol{\tau}} \tag{79}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ is the mass (inertia) matrix, which is symmetric positive definite, $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^n$ is the Coriolis and centrifugal forces, $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ is the gravitational forces, i.e., $\partial U / \partial \boldsymbol{q}$, $\boldsymbol{S} \in \mathbb{R}^{n_a \times n}$ is the selection matrix with $\boldsymbol{S} = \boldsymbol{I}_n$ for fixed base, and $\boldsymbol{\tau} \in \mathbb{R}^{n_a}$ is the actuator torques/forces.

3.5.1 Coriolis Matrix and Christoffel Symbols

The Coriolis/centrifugal term can be written as $\boldsymbol{b} = \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}$, where the **Coriolis matrix** $\boldsymbol{C}$ is defined using Christoffel symbols:

$$C_{ij} = \sum_{k=1}^{n} c_{ijk}\,\dot{q}_k, \qquad c_{ijk} = \frac{1}{2}\left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i}\right) \tag{80}$$

3.5.2 Properties of the Equations of Motion

The key properties of the equations of motion are:

- $\boldsymbol{M}(\boldsymbol{q})$ is symmetric and positive definite for all $\boldsymbol{q}$.
- $\boldsymbol{M}(\boldsymbol{q})$ is bounded: $m_{\min}\boldsymbol{I} \preceq \boldsymbol{M}(\boldsymbol{q}) \preceq m_{\max}\boldsymbol{I}$.
- $\dot{\boldsymbol{M}} - 2\boldsymbol{C}$ is **skew-symmetric**, which is crucial for Lyapunov-based stability proofs of control laws: $\dot{\boldsymbol{q}}^\top(\dot{\boldsymbol{M}} - 2\boldsymbol{C})\,\dot{\boldsymbol{q}} = 0 \quad \forall\,\dot{\boldsymbol{q}}$

3.5.3 Physical Interpretation of Each Term

- $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}$: inertial forces. Depends on configuration $\boldsymbol{q}$ because the effective inertia changes with pose, e.g., an extended arm has more rotational inertia than a tucked arm.
- $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$: velocity-dependent forces that arise from the *non-inertial* nature of the moving reference frames.
  1. **Coriolis forces**: proportional to products of *different* joint velocities $\dot{q}_i\dot{q}_j$ with $i \neq j$, arising from the interaction between rotations of different links.
  2. **Centrifugal forces**: proportional to $\dot{q}_i^2$, arising from single joints rotating, i.e., the "outward push" felt on a merry-go-round.
- $\boldsymbol{g}(\boldsymbol{q}) = \partial U/\partial \boldsymbol{q}$: gravitational forces projected into joint space, depending only on configuration, not velocity.
- $\boldsymbol{S}^\top\boldsymbol{\tau}$: actuator forces/torques. For a fixed-base fully actuated robot, $\boldsymbol{S} = \boldsymbol{I}$ and all joints are directly controlled.

3.5.4 Forward and Inverse Dynamics

The EOM, i.e., Eq. (79), can be used in two directions. **Inverse dynamics**, i.e., given motion, find torques $\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q})$. This is used in computed torque control, trajectory planning, and force estimation, and is efficiently computed by the recursive Newton–Euler algorithm in $O(n)$. **Forward dynamics**, i.e., given torques, find accelerations $\ddot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})^{-1}(\boldsymbol{\tau} - \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{g}(\boldsymbol{q}))$. This is used in simulation, model-based RL, and trajectory optimization, and requires solving a linear system or inverting $\boldsymbol{M}$, with complexity $O(n^3)$ in general.

3.5.5 Energy Considerations

The total mechanical energy $E = T + U$ satisfies $\dot{E} = \dot{\boldsymbol{q}}^\top\boldsymbol{\tau}$, which follows from the skew-symmetry of $\dot{\boldsymbol{M}} - 2\boldsymbol{C}$. If no external torques are applied, i.e., $\boldsymbol{\tau} = \boldsymbol{0}$, energy is conserved, meaning that $\dot{E} = 0$. This is the basis for **passivity-based control**: the robot is a passive system, i.e., it cannot generate energy, and controllers can exploit this structure.

3.5.6 Linearization of the EOM

At an equilibrium point $(\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0 = \boldsymbol{0})$ where $\boldsymbol{g}(\boldsymbol{q}_0) = \boldsymbol{S}^\top\boldsymbol{\tau}_0$, with perturbations $\delta\boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_0$:

$$\boldsymbol{M}(\boldsymbol{q}_0)\,\delta\ddot{\boldsymbol{q}} + \left.\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}_0}\delta\boldsymbol{q} = \boldsymbol{S}^\top\delta\boldsymbol{\tau} \tag{81}$$

The Coriolis term vanishes because it is quadratic in $\dot{\boldsymbol{q}}$. This linearized model is useful for stability analysis, i.e., eigenvalue analysis of the linearized system, linear controller design such as LQR, H_∞, and pole placement, and small-signal analysis around operating points.

3.5.7 State-Space Form

With state $\boldsymbol{z} = [\boldsymbol{q}^\top, \dot{\boldsymbol{q}}^\top]^\top \in \mathbb{R}^{2n}$, the EOM becomes a first-order ODE:

$$\dot{\boldsymbol{z}} = \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{M}^{-1}(\boldsymbol{S}^\top \boldsymbol{\tau} - \boldsymbol{b} - \boldsymbol{g}) \end{bmatrix} = \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\tau}) \tag{82}$$

This is the form used in simulation, e.g., numerical integration with Euler or RK4, and in model-based reinforcement learning as the "environment dynamics".

3.6 Dynamic Control

3.6.1 Joint-Space PD Control with Gravity Compensation

Joint-space PD control with gravity compensation is the simplest model-based controller:

$$\boxed{\boldsymbol{\tau} = \boldsymbol{K}_p\,(\boldsymbol{q}_d - \boldsymbol{q}) - \boldsymbol{K}_d\,\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q})} \tag{83}$$

where $\boldsymbol{K}_p, \boldsymbol{K}_d \in \mathbb{R}^{n\times n}$ are positive definite gain matrices, typically diagonal. Without gravity compensation, steady-state error exists because the robot sags under gravity. The **stability proof** uses a Lyapunov argument. The Lyapunov candidate is $V = \frac{1}{2}\dot{\boldsymbol{q}}^\top \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} + \frac{1}{2}\boldsymbol{e}^\top \boldsymbol{K}_p\,\boldsymbol{e}$, where $\boldsymbol{e} = \boldsymbol{q}_d - \boldsymbol{q}$. Then $V > 0$ and $\dot{V} = -\dot{\boldsymbol{q}}^\top \boldsymbol{K}_d\,\dot{\boldsymbol{q}} \le 0$ using the skew-symmetry of $\dot{\boldsymbol{M}} - 2\boldsymbol{C}$. By LaSalle's invariance principle, $\dot{\boldsymbol{q}} \to \boldsymbol{0}$ and $\boldsymbol{q} \to \boldsymbol{q}_d$.

3.6.2 Computed Torque / Inverse Dynamics Control

Computed torque control, also called inverse dynamics control, cancels the nonlinear dynamics, achieving *exact linearization*:

$$\boxed{\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\,(\ddot{\boldsymbol{q}}_d + \boldsymbol{K}_d\,\dot{\boldsymbol{e}} + \boldsymbol{K}_p\,\boldsymbol{e}) + \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q})} \tag{84}$$

where $\boldsymbol{e} = \boldsymbol{q}_d - \boldsymbol{q}$ is the tracking error. Substituting into the EOM, i.e., Eq. (79), with $\boldsymbol{S} = \boldsymbol{I}$:

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}} + \boldsymbol{b} + \boldsymbol{g} = \boldsymbol{M}(\ddot{\boldsymbol{q}}_d + \boldsymbol{K}_d\dot{\boldsymbol{e}} + \boldsymbol{K}_p\boldsymbol{e}) + \boldsymbol{b} + \boldsymbol{g}, \quad \ddot{\boldsymbol{e}} + \boldsymbol{K}_d\,\dot{\boldsymbol{e}} + \boldsymbol{K}_p\,\boldsymbol{e} = \boldsymbol{0} \tag{85}$$

This is a linear, decoupled error dynamics. Choosing $\boldsymbol{K}_p = \omega_n^2\,\boldsymbol{I}$ and $\boldsymbol{K}_d = 2\zeta\omega_n\,\boldsymbol{I}$ gives critically damped response with $\zeta = 1$. This approach requires accurate model knowledge, as model errors lead to imperfect cancellation. For **robustness to model errors**, the estimated model $\hat{\boldsymbol{M}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{g}}$ (with hats denoting estimates) gives the applied torque $\boldsymbol{\tau} = \hat{\boldsymbol{M}}(\ddot{\boldsymbol{q}}_d + \boldsymbol{K}_d\dot{\boldsymbol{e}} + \boldsymbol{K}_p\boldsymbol{e}) + \hat{\boldsymbol{b}} + \hat{\boldsymbol{g}}$, resulting in the error dynamics:

$$\boldsymbol{M}\ddot{\boldsymbol{e}} + \boldsymbol{M}\boldsymbol{K}_d\dot{\boldsymbol{e}} + \boldsymbol{M}\boldsymbol{K}_p\boldsymbol{e} = (\boldsymbol{M} - \hat{\boldsymbol{M}})(\ddot{\boldsymbol{q}}_d + \boldsymbol{K}_d\dot{\boldsymbol{e}} + \boldsymbol{K}_p\boldsymbol{e}) + (\boldsymbol{b} - \hat{\boldsymbol{b}}) + (\boldsymbol{g} - \hat{\boldsymbol{g}}) \tag{86}$$

The right-hand side represents the effect of model errors. If $\hat{\boldsymbol{M}} \approx \boldsymbol{M}$, the error dynamics are approximately linear and the system remains stable for sufficiently large gains. The simplest variant is **gravity compensation only** $\boldsymbol{\tau} = \boldsymbol{K}_p\boldsymbol{e} + \boldsymbol{K}_d\dot{\boldsymbol{e}} + \hat{\boldsymbol{g}}(\boldsymbol{q})$. This requires only the gravity model, i.e., no $\boldsymbol{M}$ or $\boldsymbol{b}$, making it more robust to model errors while still achieving zero steady-state error for regulation, i.e., set-point control.

3.6.3 Impedance Control

Impedance control, instead of tracking a trajectory exactly, specifies a desired *dynamic behavior*, i.e., spring-damper, at the end-effector $\boldsymbol{\tau} = \boldsymbol{J}^\top(\boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x}) - \boldsymbol{D}\,\dot{\boldsymbol{x}}) + \boldsymbol{g}(\boldsymbol{q})$. The robot behaves as a virtual spring-damper system in task space. Unlike computed torque, impedance control does not cancel the dynamics, i.e., it allows compliant interaction with the environment. The **desired impedance** specifies that the end-effector satisfies $\boldsymbol{\Lambda}_d\,\ddot{\boldsymbol{e}}_x + \boldsymbol{D}_d\,\dot{\boldsymbol{e}}_x + \boldsymbol{K}_s\,\boldsymbol{e}_x = \boldsymbol{f}^{\text{ext}}$ where $\boldsymbol{\Lambda}_d$, $\boldsymbol{D}_d$, $\boldsymbol{K}_s$ are the desired inertia, damping, and stiffness, respectively. This allows the robot to be stiff in some directions and compliant in others, which is essential for contact tasks, e.g., polishing and insertion. The key trade-off is that impedance control sacrifices tracking accuracy for safe interaction. In contrast, position/force hybrid control explicitly partitions task-space directions into position-controlled and force-controlled subspaces.

3.6.4 Task-Space Inverse Dynamics

The **task-space (operational space) dynamics** are derived from the joint-space EOM, i.e., Eq. (79), with $\boldsymbol{S} = \boldsymbol{I}$ and the kinematic relation $\dot{\boldsymbol{x}} = \boldsymbol{J}\,\dot{\boldsymbol{q}}$. The **task-space inertia matrix** is $\boldsymbol{\Lambda}(\boldsymbol{q}) = \left(\boldsymbol{J}\,\boldsymbol{M}^{-1}\,\boldsymbol{J}^\top\right)^{-1}$. The **task-space bias forces** are $\boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{\Lambda}\Big(\boldsymbol{J}\,\boldsymbol{M}^{-1}\,\boldsymbol{b} - \dot{\boldsymbol{J}}\,\dot{\boldsymbol{q}}\Big)$. The **task-space gravity** is $\boldsymbol{p}(\boldsymbol{q}) = \boldsymbol{\Lambda}\,\boldsymbol{J}\,\boldsymbol{M}^{-1}\,\boldsymbol{g}$. Then the task-space dynamics become:

$$\boxed{\boldsymbol{\Lambda}\,\ddot{\boldsymbol{x}} + \boldsymbol{\mu} + \boldsymbol{p} = \boldsymbol{f}} \tag{87}$$

where $\boldsymbol{f}$ is the task-space force and the corresponding joint torques are $\boldsymbol{\tau} = \boldsymbol{J}^\top\,\boldsymbol{f}$. The **task-space inverse dynamics controller** is $\boldsymbol{f} = \boldsymbol{\Lambda}(\ddot{\boldsymbol{x}}_d + \boldsymbol{K}_d\,\dot{\boldsymbol{e}}_x + \boldsymbol{K}_p\,\boldsymbol{e}_x) + \boldsymbol{\mu} + \boldsymbol{p}$ where $\boldsymbol{e}_x = \boldsymbol{x}_d - \boldsymbol{x}$, achieving linear error dynamics in task space, i.e., $\ddot{\boldsymbol{e}}_x + \boldsymbol{K}_d\dot{\boldsymbol{e}}_x + \boldsymbol{K}_p\boldsymbol{e}_x = \boldsymbol{0}$. For **redundant** robots, adding null-space torques yields $\boldsymbol{\tau} = \boldsymbol{J}^\top\,\boldsymbol{f} + (\boldsymbol{I} - \boldsymbol{J}^\top\,\bar{\boldsymbol{J}}^\top)\,\boldsymbol{\tau}_0$ where $\bar{\boldsymbol{J}} = \boldsymbol{M}^{-1}\boldsymbol{J}^\top\boldsymbol{\Lambda}$ is the **dynamically consistent pseudoinverse** and $\boldsymbol{\tau}_0$ achieves a secondary objective in the null space. The standard kinematic pseudoinverse $\boldsymbol{J}^+ = \boldsymbol{J}^\top(\boldsymbol{J}\boldsymbol{J}^\top)^{-1}$ minimizes $\|\dot{\boldsymbol{q}}\|^2$, whereas the dynamically consistent pseudoinverse $\bar{\boldsymbol{J}} = \boldsymbol{M}^{-1}\boldsymbol{J}^\top\boldsymbol{\Lambda}$ instead minimizes the kinetic energy $\frac{1}{2}\dot{\boldsymbol{q}}^\top\boldsymbol{M}\dot{\boldsymbol{q}}$, which is physically more meaningful. This ensures that null-space motions do not generate task-space accelerations, i.e., they are "dynamically decoupled" from the task. The **task-space inertia $\boldsymbol{\Lambda}$** has the following properties:

- Symmetric positive definite, like $\boldsymbol{M}$ in joint space
- Reflects the effective inertia felt at the end-effector
- Configuration-dependent: $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}(\boldsymbol{q})$
- At singularities, $\boldsymbol{\Lambda} \to \infty$ in certain directions, i.e., infinite apparent inertia, meaning it is impossible to accelerate

The **derivation of task-space dynamics** starts from $\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{b} + \boldsymbol{g} = \boldsymbol{\tau}$ and $\ddot{\boldsymbol{x}} = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$:

$$\begin{aligned}
\ddot{\boldsymbol{q}} &= \boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{b} - \boldsymbol{g}) \\
\ddot{\boldsymbol{x}} &= \boldsymbol{J}\boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{b} - \boldsymbol{g}) + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} = \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^\top\,\boldsymbol{f} - \boldsymbol{J}\boldsymbol{M}^{-1}(\boldsymbol{b} + \boldsymbol{g}) + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} \quad (\text{setting } \boldsymbol{\tau} = \boldsymbol{J}^\top\boldsymbol{f}) \\
\boldsymbol{\Lambda}^{-1}\boldsymbol{f} &= \ddot{\boldsymbol{x}} + \boldsymbol{J}\boldsymbol{M}^{-1}(\boldsymbol{b} + \boldsymbol{g}) - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}
\end{aligned} \tag{88}$$

Multiplying by $\boldsymbol{\Lambda}$ and rearranging gives Eq. (87).

3.7 Floating Base Dynamics

For robots with an unactuated base, e.g., legged robots and humanoids, the **generalized coordinates** are:

$$\boldsymbol{q} = \begin{bmatrix}\boldsymbol{q}_b \\ \boldsymbol{q}_j\end{bmatrix}, \quad \boldsymbol{q}_b = \begin{bmatrix}\boldsymbol{r}_{IB} \\ \boldsymbol{\Phi}\end{bmatrix} \in \mathbb{R}^6, \quad \boldsymbol{q}_j \in \mathbb{R}^{n_j} \tag{89}$$

The **selection matrix** reflects that only the joints are actuated, not the base:

$$\boldsymbol{S} = \begin{bmatrix}\boldsymbol{0}_{n_j\times 6} & \boldsymbol{I}_{n_j}\end{bmatrix} \quad \Rightarrow \quad \boldsymbol{S}^\top\,\boldsymbol{\tau}_j = \begin{bmatrix}\boldsymbol{0}_6 \\ \boldsymbol{\tau}_j\end{bmatrix} \tag{90}$$

The **equations of motion with contacts** are:

$$\boxed{\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{b}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{S}^\top \boldsymbol{\tau}_j + \sum_k \boldsymbol{J}_{c_k}^\top \boldsymbol{\lambda}_k} \tag{91}$$

where $\boldsymbol{\lambda}_k \in \mathbb{R}^3$ is the contact force at contact point k and $\boldsymbol{J}_{c_k}$ is the contact Jacobian mapping generalized velocities to contact point velocity. The first 6 rows of Eq. (91) correspond to the unactuated base, i.e., Newton–Euler equations for the whole-body center of mass $\boldsymbol{M}_b\,\ddot{\boldsymbol{q}} + \boldsymbol{b}_b + \boldsymbol{g}_b = \sum_k \boldsymbol{J}_{c_k,b}^\top \boldsymbol{\lambda}_k$. This enforces that only contact forces can accelerate the base, as there are no direct actuators on the floating base. The **contact constraints** require that during stance the contact point has zero velocity $\boldsymbol{J}_{c_k}\,\dot{\boldsymbol{q}} = \boldsymbol{0} \Rightarrow \boldsymbol{J}_{c_k}\,\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{c_k}\,\dot{\boldsymbol{q}} = \boldsymbol{0}$. The contact forces $\boldsymbol{\lambda}_k$ are constraint forces that must satisfy:

- Unilateral constraint: normal force $\lambda_n \geq 0$, i.e., ground can only push
- Friction cone: $\sqrt{\lambda_x^2 + \lambda_y^2} \leq \mu\,\lambda_n$, i.e., Coulomb friction where μ is the friction coefficient

3.7.1 Block Structure of Floating Base EOM

Partitioning $\boldsymbol{q} = [\boldsymbol{q}_b^\top, \boldsymbol{q}_j^\top]^\top$ into base (b) and joint (j) components, the EOM becomes:

$$\begin{bmatrix} \boldsymbol{M}_{bb} & \boldsymbol{M}_{bj} \\ \boldsymbol{M}_{jb} & \boldsymbol{M}_{jj} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_b \\ \ddot{\boldsymbol{q}}_j \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_b \\ \boldsymbol{b}_j \end{bmatrix} + \begin{bmatrix} \boldsymbol{g}_b \\ \boldsymbol{g}_j \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\tau}_j \end{bmatrix} + \begin{bmatrix} \boldsymbol{J}_{c,b}^\top \\ \boldsymbol{J}_{c,j}^\top \end{bmatrix} \boldsymbol{\lambda} \tag{92}$$

The top block, i.e., base dynamics with 6 equations, has no actuation, so the base can only be accelerated through contact forces or gravity.

3.7.2 Inverse Dynamics for Floating Base

Given desired joint accelerations $\ddot{\boldsymbol{q}}_j$ and contact constraints, the joint torques $\boldsymbol{\tau}_j$ and contact forces $\boldsymbol{\lambda}$ are determined as follows. From the contact constraint $\boldsymbol{J}_c\ddot{\boldsymbol{q}} = -\dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}}$ and the EOM, i.e., Eq. (92), the system of equations is:

$$\begin{bmatrix} \boldsymbol{M} & -\boldsymbol{J}_c^\top \\ \boldsymbol{J}_c & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{S}^\top\boldsymbol{\tau}_j - \boldsymbol{b} - \boldsymbol{g} \\ -\dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} \end{bmatrix} \tag{93}$$

This is a saddle-point system, i.e., KKT system, which also appears in constrained optimization.

3.7.3 Whole-Body Control

For floating-base robots, a common approach is **whole-body control (WBC)**, which formulates the control problem as an optimization:

$$\min_{\ddot{\boldsymbol{q}},\boldsymbol{\tau}_j,\boldsymbol{\lambda}} \quad \text{tracking cost} \quad \text{s.t.} \quad \boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{b} + \boldsymbol{g} = \boldsymbol{S}^\top\boldsymbol{\tau}_j + \boldsymbol{J}_c^\top\boldsymbol{\lambda}, \quad \boldsymbol{J}_c\ddot{\boldsymbol{q}} = -\dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}}, \quad \boldsymbol{\lambda} \in \text{friction cone} \tag{94}$$

This is a quadratic program, i.e., QP, solved at each control step.

3.7.4 Centroidal Dynamics

For the *whole system*, Newton's and Euler's laws give:

$$m_{\text{tot}}\,\boldsymbol{a}_{\text{CoM}} = \sum_k \boldsymbol{\lambda}_k + m_{\text{tot}}\,\boldsymbol{g}_0, \quad \dot{\boldsymbol{h}}_G = \sum_k \boldsymbol{r}_{Gc_k} \times \boldsymbol{\lambda}_k \tag{95}$$

where $\boldsymbol{g}_0 \in \mathbb{R}^3$ is the gravitational acceleration vector, e.g., $\boldsymbol{g}_0 = [0,\, 0,\, -g]^\top$ (not to be confused with the generalized gravity force $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ in Eq. (79)), m_{tot} is the total mass, $\boldsymbol{a}_{\text{CoM}}$ is the center-of-mass acceleration, $\boldsymbol{h}_G$ is the centroidal angular momentum, i.e., angular momentum about the CoM, and $\boldsymbol{r}_{Gc_k}$ is the vector from the CoM to contact point k. These 6 equations are the *centroidal dynamics*. They are physically equivalent to the top 6 rows of the floating-base EOM, i.e., Eq. (92). Both express whole-body momentum balance, but the former references the CoM while the latter references the

base-frame origin, related by the coordinate transformation. The key insight is that the centroidal dynamics depend only on external forces, i.e., contacts and gravity, and are independent of internal joint torques. This makes them useful for planning, e.g., walking pattern generation, before solving for joint-level motions. The **centroidal momentum matrix** $\boldsymbol{A}_G(\boldsymbol{q}) \in \mathbb{R}^{6\times(6+n_j)}$ maps generalized velocities to the centroidal momentum:

$$\boldsymbol{h}_G^{\text{full}} = \begin{bmatrix} m_{\text{tot}}\,\boldsymbol{v}_{\text{CoM}} \\ \boldsymbol{h}_G \end{bmatrix} = \boldsymbol{A}_G(\boldsymbol{q})\,\dot{\boldsymbol{q}} \tag{96}$$

where $\boldsymbol{h}_G^{\text{full}}$ stacks the linear and angular momenta about the CoM. $\boldsymbol{A}_G$ depends on the configuration $\boldsymbol{q}$ but *not* on $\dot{\boldsymbol{q}}$, and its time derivative gives $\dot{\boldsymbol{h}}_G^{\text{full}} = \boldsymbol{A}_G\ddot{\boldsymbol{q}} + \dot{\boldsymbol{A}}_G\dot{\boldsymbol{q}}$. It can be computed from the mass matrix: $\boldsymbol{A}_G = \boldsymbol{X}_G^{\top}\boldsymbol{M}$, where $\boldsymbol{X}_G$ is the coordinate transform from generalized coordinates to the CoM frame. The **single rigid body model (SRBM)** approximates the entire robot as one rigid body with total mass m_{tot} and a constant (configuration-averaged) rotational inertia $\bar{\boldsymbol{\Theta}}$:

$$\boxed{m_{\text{tot}}\,\ddot{\boldsymbol{r}}_c = \sum_k \boldsymbol{\lambda}_k + m_{\text{tot}}\,\boldsymbol{g}_0, \quad \bar{\boldsymbol{\Theta}}\,\dot{\boldsymbol{\omega}} + \boldsymbol{\omega}\times(\bar{\boldsymbol{\Theta}}\,\boldsymbol{\omega}) = \sum_k \boldsymbol{r}_{cc_k}\times\boldsymbol{\lambda}_k} \tag{97}$$

where $\boldsymbol{r}_c$ is the CoM position and $\boldsymbol{r}_{cc_k}$ is the vector from CoM to contact k. The SRBM has only 6 DoF (CoM position + orientation) and is *independent of joint angles*, making it efficient for MPC planning (see Section 4.4). When the angular velocity is small, the angular dynamics linearize, and the resulting OCP becomes a **convex QP** that can be solved at kHz rates. The **zero-moment point (ZMP)** for a humanoid on a flat ground is the point on the ground where the net ground reaction force produces zero moment about the horizontal axes. The robot remains balanced if and only if the ZMP lies within the support polygon, i.e., the convex hull of the contact points.

3.7.5 Summary: Floating Base vs. Fixed Base

	Fixed Base	**Floating Base**
DoF	n (joint angles)	$6+n_j$
Actuated DoF	n	n_j (no base actuation)
Selection matrix $\boldsymbol{S}$	$\boldsymbol{I}_n$	$[\boldsymbol{0}, \boldsymbol{I}_{n_j}]$
Contact forces	Not needed	Essential ($\boldsymbol{J}_c^{\top}\boldsymbol{\lambda}$)
Base motion	Fixed	Determined by contacts
Underactuated?	No	Yes (6 unactuated DoF)

3.7.6 Contact Models

The **rigid contact** model, used in most control formulations, assumes zero penetration and perfectly rigid surfaces. Contact forces are determined as constraint forces, i.e., Lagrange multipliers $\boldsymbol{\lambda}$. The **compliant contact** model, used in many simulators, represents the contact surface as a spring-damper: $\lambda_n = k\,\delta + d\,\dot{\delta}$ for penetration depth $\delta > 0$, where k is stiffness and d is damping. Common **friction models** include:

- **Coulomb (dry) friction:** $\|\boldsymbol{f}_t\| \le \mu\,\lambda_n$, where $\boldsymbol{f}_t$ is the tangential force. During sliding, $\boldsymbol{f}_t = -\mu\,\lambda_n\,\hat{\boldsymbol{v}}_t$, i.e., opposes sliding direction. During sticking, $\|\boldsymbol{f}_t\| < \mu\,\lambda_n$ and $\boldsymbol{v}_t = \boldsymbol{0}$.
- **Viscous friction:** $\boldsymbol{f}_t = -d_t\,\boldsymbol{v}_t$, which is linear in velocity and easier to simulate.

The friction cone $\sqrt{f_x^2 + f_y^2} \le \mu\,\lambda_n$ is a second-order cone constraint. For computational efficiency, it is often approximated by a **friction pyramid**, i.e., polyhedral approximation with linear constraints.

3.7.7 Contact Switches and Impact Collisions

A hard contact model subdivides the dynamics into phases separated by **impact events**, i.e., instantaneous changes in the contact situation. Integrating the EOM over an infinitesimally short impact at time t_0 eliminates all finite terms ($\boldsymbol{b}$, $\boldsymbol{g}$, $\boldsymbol{\tau}$), leaving the **impulse balance**:

$$\boxed{\boldsymbol{M}\left(\dot{\boldsymbol{q}}^{+} - \dot{\boldsymbol{q}}^{-}\right) = \boldsymbol{J}_c^{\top}\,\mathcal{F}_c} \tag{98}$$

where $\dot{\boldsymbol{q}}^-$ and $\dot{\boldsymbol{q}}^+$ are the pre- and post-impact generalized velocities and $\mathcal{F}_c$ is the impulsive contact force. For a perfectly inelastic collision, i.e., coefficient of restitution $\varepsilon = 0$, the contact point comes to rest: $\dot{\boldsymbol{r}}_c^+ = \boldsymbol{J}_c\,\dot{\boldsymbol{q}}^+ = \boldsymbol{0}$. The **contact-point inertia** (also called operational space or end-effector inertia at the contact) is $\boldsymbol{\Lambda}_c = \left(\boldsymbol{J}_c\,\boldsymbol{M}^{-1}\,\boldsymbol{J}_c^\top\right)^{-1}$. The impulsive force is $\mathcal{F}_c = -\boldsymbol{\Lambda}_c\,\dot{\boldsymbol{r}}_c^-$ and the velocity jump in generalized coordinates is $\Delta\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}^+ - \dot{\boldsymbol{q}}^- = -\boldsymbol{M}^{-1}\,\boldsymbol{J}_c^\top\,\boldsymbol{\Lambda}_c\,\boldsymbol{J}_c\,\dot{\boldsymbol{q}}^-$. Equivalently, the post-impact velocity $\dot{\boldsymbol{q}}^+ = \boldsymbol{N}_c\,\dot{\boldsymbol{q}}^-$ is obtained by projecting $\dot{\boldsymbol{q}}^-$ onto the contact-consistent manifold via the **support null-space projector** $\boldsymbol{N}_c = \boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{J}_c^\top\left(\boldsymbol{J}_c\boldsymbol{M}^{-1}\boldsymbol{J}_c^\top\right)^{-1}\boldsymbol{J}_c$. For the perfectly inelastic case, i.e., $\varepsilon = 0$, the energy lost during impact is:

$$E_{\text{loss}} = \tfrac{1}{2}\,\Delta\dot{\boldsymbol{q}}^\top\,\boldsymbol{M}\,\Delta\dot{\boldsymbol{q}} = \tfrac{1}{2}\,\dot{\boldsymbol{r}}_c^{-\top}\,\boldsymbol{\Lambda}_c\,\dot{\boldsymbol{r}}_c^- \geq 0 \tag{99}$$

This identity holds because $\boldsymbol{I} - \boldsymbol{N}_c$ is an $\boldsymbol{M}$-orthogonal projector. For partial restitution ($0 < \varepsilon \leq 1$), the post-impact contact velocity satisfies $\dot{\boldsymbol{r}}_c^+ = -\varepsilon\,\dot{\boldsymbol{r}}_c^-$.

4 Optimal Control and Trajectory Optimization

This section covers the optimization methods used to generate motions over the robot dynamics model from Section 3, i.e., Eq. (79) and Eq. (91). Note that $\boldsymbol{x}$ denotes the **state** vector, e.g., $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$, and $\boldsymbol{u}$ the control input, as is standard in optimal control. This differs from Section 2, where $\boldsymbol{x}$ denotes the task-space position.

4.1 Problem Formulation

The continuous-time **optimal control problem (OCP)** is:

$$\boxed{\min_{\boldsymbol{u}(\cdot)} \quad \int_0^T \ell(\boldsymbol{x}(t), \boldsymbol{u}(t))\,dt + \phi(\boldsymbol{x}(T)) \quad \text{s.t.} \quad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \quad \boldsymbol{x}(t) \in \mathcal{X}, \quad \boldsymbol{u}(t) \in \mathcal{U}} \tag{100}$$

where $\ell(\boldsymbol{x}, \boldsymbol{u})$ is the running cost, $\phi(\boldsymbol{x}(T))$ is the terminal cost, and $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ represents the system dynamics. For rigid-body robots, $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$ and the dynamics $\boldsymbol{f}$ come from the EOM, i.e., Eq. (79): $\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{S}^\top\boldsymbol{\tau} - \boldsymbol{b} - \boldsymbol{g})$, with $\boldsymbol{u} = \boldsymbol{\tau}$. As a concrete example, consider a robot arm tracking a desired joint trajectory $\boldsymbol{q}^{\text{des}}(t)$ with minimal effort. The running cost penalizes both tracking error and control effort: $\ell(\boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{q} - \boldsymbol{q}^{\text{des}}\|_{\boldsymbol{Q}}^2 + \frac{1}{2}\|\boldsymbol{u}\|_{\boldsymbol{W}}^2$, where $\boldsymbol{Q} \succeq 0$ and $\boldsymbol{W} \succ 0$ are weight matrices. Note that $\boldsymbol{W}$ is used instead of the conventional R to avoid conflict with the rotation matrix $\boldsymbol{R}$. The terminal cost penalizes the final-state error: $\phi(\boldsymbol{x}(T)) = \frac{1}{2}\|\boldsymbol{q}(T) - \boldsymbol{q}^{\text{des}}(T)\|_{\boldsymbol{Q}_f}^2$, where $\boldsymbol{Q}_f \succeq 0$ is typically chosen larger than $\boldsymbol{Q}$ to strongly enforce the goal. The control input is $\boldsymbol{u} = \boldsymbol{\tau}$ (joint torques).

For floating-base systems with contacts, the dynamics switch at contact transitions, making it a **hybrid dynamical system**. Namely, each phase has its own dynamics $\boldsymbol{f}_i$ and contact constraints, and the contact schedule, e.g., which feet are in stance, defines the phase sequence.

4.2 Transcription Methods

To solve the OCP numerically, it is transcribed into a finite-dimensional nonlinear program (NLP). The main approaches differ in what is discretized. All methods enforce the *same* underlying continuous dynamics $\boldsymbol{f}$ from Eq. (100). Namely, they differ only in how this constraint is represented in the NLP: $\boldsymbol{f}_d$ in direct shooting wraps $\boldsymbol{f}$ inside a numerical integrator, e.g., Euler or RK4, while $\boldsymbol{c}_k$ in direct collocation evaluates $\boldsymbol{f}$ at quadrature points and enforces consistency as an equality constraint.

4.2.1 Direct Shooting

Direct shooting discretizes only the controls $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}$. The states are obtained by forward simulation $\boldsymbol{x}_{k+1} = \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k)$, where $\boldsymbol{f}_d$ is the discrete-time dynamics, e.g., Euler or Runge–Kutta

integration of $\boldsymbol{f}$ [4]. The NLP has decision variables $\boldsymbol{u}_{0:N-1}$ only:

$$\min_{\boldsymbol{u}_{0:N-1}} \sum_{k=0}^{N-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \phi(\boldsymbol{x}_N) \quad \text{where} \quad \boldsymbol{x}_{k+1} = \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k), \;\; \boldsymbol{x}_0 \text{ given} \tag{101}$$

- **Advantages:** Small NLP (controls only), simple implementation
- **Disadvantages:** Sequential dynamics evaluation (no parallelism); unstable systems cause exponential sensitivity to early controls

4.2.2 Direct Collocation

Direct collocation discretizes both states $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_N$ and controls $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}$ as decision variables. The dynamics are enforced as *equality constraints* at collocation points (e.g., midpoints using Hermite–Simpson or Gauss–Legendre quadrature):

$$\min_{\boldsymbol{x}_{0:N},\, \boldsymbol{u}_{0:N-1}} \sum_{k=0}^{N-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \phi(\boldsymbol{x}_N) \quad \text{s.t.} \quad \boldsymbol{c}_k(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}, \boldsymbol{u}_k) = \boldsymbol{0} \tag{102}$$

where $\boldsymbol{c}_k$ denotes the collocation defect constraint, e.g., for Hermite–Simpson or trapezoidal schemes. Unlike direct shooting, both $\boldsymbol{x}_{0:N}$ and $\boldsymbol{u}_{0:N-1}$ are free decision variables, with dynamics enforced as equality constraints rather than being satisfied by construction through rollout.

- **Advantages:** Sparse KKT structure (block-banded), exploitable by interior-point or SQP solvers; larger convergence basin; dynamics evaluated in parallel
- **Disadvantages:** Larger NLP; dynamics defects can yield physically inconsistent intermediate iterates

4.2.3 Multiple Shooting

Multiple shooting is a hybrid of the above two methods. The horizon is divided into segments, dynamics are forward-simulated within each segment, and **continuity constraints** are imposed at segment boundaries. This balances the small variable count of shooting with the parallelism and convergence properties of collocation.

4.2.4 Contact-Implicit Trajectory Optimization

In the methods above, the **contact schedule**, i.e., which contacts are active at each time step, must be specified a priori. **Contact-implicit** formulations instead let the optimizer discover the contact sequence by including contact forces and complementarity conditions as part of the NLP. The decision variables are augmented to $(\boldsymbol{x}_{0:N}, \boldsymbol{u}_{0:N-1}, \boldsymbol{\lambda}_{0:N-1})$, and the following constraints are added at each time step:

$$\boxed{d_i(\boldsymbol{q}_k) \geq 0, \quad \lambda_{n,i,k} \geq 0, \quad d_i(\boldsymbol{q}_k)\, \lambda_{n,i,k} = 0} \tag{103}$$

where $d_i(\boldsymbol{q})$ is the signed distance of contact point i from the surface and $\lambda_{n,i}$ is the corresponding normal force. This is a **mathematical program with complementarity constraints (MPCC)**. Friction is enforced via the friction cone $\|\boldsymbol{\lambda}_{t,i}\| \leq \mu\, \lambda_{n,i}$ with the complementarity $\|\boldsymbol{v}_{t,i}\|\, (\mu\, \lambda_{n,i} - \|\boldsymbol{\lambda}_{t,i}\|) = 0$ (sliding only when at the friction cone boundary). Since the complementarity $d\, \lambda_n = 0$ violates standard constraint qualifications (LICQ fails at the solution), common relaxation strategies include:

- **Relaxed complementarity:** replace $d\, \lambda_n = 0$ with $d\, \lambda_n \leq \epsilon$ and anneal $\epsilon \to 0$
- **Penalty method:** add $\rho\, d\, \lambda_n$ to the cost instead of as a constraint
- **Smooth contact models:** use a differentiable approximation $\lambda_n = k\, \max(0, -d)^p$ to avoid the combinatorial structure entirely

Contact-implicit methods can discover non-trivial contact sequences, e.g., a robot deciding where and when to place its feet, but the resulting NLP is non-convex with many local minima and typically requires good initialization.

[4] For example, Euler integration yields $\frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta t} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \Rightarrow \boldsymbol{x}_{k+1} = \Delta t\, \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{x}_k = \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k)$.

4.3 iLQR and Differential Dynamic Programming

Iterative LQR (iLQR) and **Differential Dynamic Programming (DDP)** solve nonlinear trajectory optimization via sequential quadratic approximations, exploiting the temporal chain structure of dynamics. Given a current trajectory $(\bar{\boldsymbol{x}}_{0:N}, \bar{\boldsymbol{u}}_{0:N-1})$:

1. **Backward pass:** Linearize dynamics and quadratize cost around the current trajectory. Solve the resulting time-varying LQR problem via Riccati recursion, yielding local feedback gains $\boldsymbol{K}_k$ and feedforward corrections $\boldsymbol{d}_k$:
$$\boxed{\delta \boldsymbol{u}_k = \boldsymbol{K}_k \, \delta \boldsymbol{x}_k + \alpha \, \boldsymbol{d}_k} \tag{104}$$
2. **Forward pass:** Roll out the dynamics with the updated policy $\boldsymbol{u}_k = \bar{\boldsymbol{u}}_k + \delta \boldsymbol{u}_k$, using line search over $\alpha \in (0, 1]$.
3. Repeat until convergence.

DDP uses the full second-order expansion of the dynamics, i.e., $\boldsymbol{f}_{\boldsymbol{xx}}$ terms; **iLQR** drops them, a Gauss–Newton-like approximation, which is simpler and often sufficient. The key insight is that the Riccati recursion is equivalent to a **block-tridiagonal factorization of the KKT system** of the full NLP, exploiting the sequential time structure. This makes each iteration $O(N)$ in the horizon length, compared to $O(N^3)$ for a generic QP solver.

4.4 Model Predictive Control for Locomotion

Model predictive control (MPC) solves a finite-horizon OCP at each control step, applies the first action, and re-solves at the next step (receding horizon):

$$\boxed{\boldsymbol{u}_0^* = \arg\min_{\boldsymbol{u}_{0:N-1}} \sum_{k=0}^{N-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \phi(\boldsymbol{x}_N) \quad \text{s.t.} \quad \text{dynamics} + \text{constraints}} \tag{105}$$

Only $\boldsymbol{u}_0^*$ is applied; the problem is re-solved with updated state $\boldsymbol{x}_1$. For legged locomotion, a common architecture uses **two levels**:

1. **MPC planner:** Solves a reduced-order OCP (e.g., centroidal dynamics or the SRBM, i.e., Eq. (97)) over a longer horizon to plan center-of-mass trajectories and contact forces.
2. **Whole-body controller (WBC):** Tracks the MPC plan using the whole-body QP from Section 3.7 to compute joint torques and contact-consistent accelerations at high rate.

This separation exploits the fact that centroidal dynamics are low-dimensional (6 DoF) and independent of joint torques, while the WBC handles the full kinematics and actuation constraints.

4.5 Applications in Modern Robotics

In **model-based reinforcement learning (MBRL)**, the equations of motion $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ serve as the "world model". MPC, i.e., Eq. (105), solves a trajectory optimization problem at each control step using shooting methods, direct collocation, or differentiable simulation. In **sim-to-real transfer**, the dynamics model $\boldsymbol{M}, \boldsymbol{b}, \boldsymbol{g}$ can have uncertainties, e.g., friction, link masses, and motor dynamics. Techniques from robust control and domain randomization address the gap between simulation and reality. **Learning-based control** combines physics-based models with neural networks:

- Residual physics: $\boldsymbol{\tau} = \boldsymbol{\tau}_{\text{model}} + \boldsymbol{\tau}_{\text{NN}}$, where the model provides structure while the NN compensates errors
- Lagrangian neural networks: learn T and U as neural networks, then derive EOM automatically via the Euler–Lagrange equations
- Hamiltonian neural networks: learn the Hamiltonian $H = T + U$ and derive dynamics from Hamilton's equations

5 State Estimation

Sections 2–4 assumed perfect knowledge of the robot's state $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$. In practice, however, the full state is never directly measured: joint encoders are noisy, floating-base pose must be inferred, and

sensor readings arrive at different rates with different latencies. **State estimation** fuses noisy, partial observations into a coherent belief over the state by combining a predictive dynamics model with incoming measurements. This section introduces the Kalman filter (KF) and its nonlinear extensions (EKF), then applies them to the concrete problem of estimating the floating-base state of a legged robot from IMU and leg kinematics.

5.1 Kalman Filter and Extended Kalman Filter

The **Kalman filter** is recursive Gaussian conditioning on a Markov chain. All equations follow from the joint-Gaussian conditioning identity $p(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}\big(\hat{\boldsymbol{x}} + \boldsymbol{\Sigma}_{xz}\boldsymbol{\Sigma}_{zz}^{-1}(\boldsymbol{z} - \hat{\boldsymbol{z}}),\ \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xz}\boldsymbol{\Sigma}_{zz}^{-1}\boldsymbol{\Sigma}_{zx}\big)$. The underlying **state-space models** are:

$$\text{Linear: } \begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k + \boldsymbol{w}_k \\ \boldsymbol{z}_k = \boldsymbol{H}_k\boldsymbol{x}_k + \boldsymbol{\nu}_k \end{cases} \qquad \text{Nonlinear: } \begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{w}_k \\ \boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{\nu}_k \end{cases} \tag{106}$$

where $\boldsymbol{z}_k \in \mathbb{R}^p$ is the measurement, $\boldsymbol{A}_k$ the state transition matrix, $\boldsymbol{B}_k$ the input matrix, $\boldsymbol{H}_k$ the observation matrix, $\boldsymbol{f}$ and h are the nonlinear process and observation models [5], $\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k)$ is the process noise with covariance $\boldsymbol{Q}_k$, and $\boldsymbol{\nu}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{N}_k)$ is the measurement noise with covariance $\boldsymbol{N}_k$, both independent. Note that $\boldsymbol{N}_k$ is used for the measurement noise covariance to avoid notational conflict with the rotation matrix $\boldsymbol{R}$, while many KF references denote this $\boldsymbol{R}$. The **predict–update cycle** for both the linear KF and the EKF is:

$$\begin{aligned}
\text{Predict:}\quad & \hat{\boldsymbol{x}}_k^- = \boldsymbol{A}_{k-1}\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}_{k-1}\boldsymbol{u}_{k-1}\ \text{(KF)}, \quad \hat{\boldsymbol{x}}_k^- = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_{k-1})\ \text{(EKF)} \\
& \boldsymbol{P}_k^- = \boldsymbol{F}_{k-1}\,\boldsymbol{P}_{k-1}\,\boldsymbol{F}_{k-1}^\top + \boldsymbol{Q}_{k-1}, \quad \boldsymbol{F} = \boldsymbol{A}\ \text{(KF)}, \quad \boldsymbol{F} = \partial\boldsymbol{f}/\partial\boldsymbol{x}\big|_{\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_{k-1}}\ \text{(EKF)} \\
\text{Update:}\quad & \boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}_k^\top(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^\top + \boldsymbol{N}_k)^{-1} \\
& \hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-)\ \text{(KF)}, \quad \hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\big(\boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_k^-)\big),\ \boldsymbol{H} = \partial h/\partial\boldsymbol{x}\big|_{\hat{\boldsymbol{x}}_k^-}\ \text{(EKF)} \\
& \boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\,\boldsymbol{P}_k^-
\end{aligned} \tag{107}$$

where $\boldsymbol{P}_k$ is the state error covariance and $\boldsymbol{K}_k$ the Kalman gain. For robotics, the standard EKF has key limitations such as underestimation of covariance due to the first-order linearization, gimbal lock with Euler angles, and quaternion incompatibility with additive Gaussian noise because of a unit-norm constraint. The error-state formulation resolves these issues.

5.2 Error-State Kalman Filter

Since SO(3) is a Lie group and not a vector space, Euler angles are singular, quaternions are constrained, and rotation matrices are over-parameterized. The **error-state Kalman filter (ESKF)** resolves this by separating the state into a *nominal state* propagated through full nonlinear dynamics and a small *error state* $\delta\boldsymbol{x}$ living in the tangent space where Gaussians are valid. The error state is reset to zero after each update, keeping linearization accurate [6]. **nominal state** and **error state** are defined as:

$$\boldsymbol{x}_{\text{nom}} = [\boldsymbol{r},\ \boldsymbol{v},\ \bar{q},\ \boldsymbol{b}_a,\ \boldsymbol{b}_g]^\top \in \mathbb{R}^3\times\mathbb{R}^3\times S^3\times\mathbb{R}^3\times\mathbb{R}^3, \qquad \delta\boldsymbol{x} = [\delta\boldsymbol{r},\ \delta\boldsymbol{v},\ \delta\boldsymbol{\theta},\ \delta\boldsymbol{b}_a,\ \delta\boldsymbol{b}_g]^\top \in \mathbb{R}^{15} \tag{108}$$

where $(\boldsymbol{r}, \boldsymbol{v})$ are world-frame position/velocity, $\bar{q}$ is the unit quaternion orientation, $(\boldsymbol{b}_a, \boldsymbol{b}_g)$ are IMU biases, and $\delta\boldsymbol{\theta} \in \mathbb{R}^3$ is a small rotation vector (axis-angle, *not* quaternion difference). The true state combines nominal and error via the **composition operator** ($\boxplus$): additive for $\boldsymbol{r}, \boldsymbol{v}, \boldsymbol{b}_a, \boldsymbol{b}_g$; multiplicative for orientation:

$$\boxed{\bar{q}_{\text{true}} = \bar{q}_{\text{nom}} \otimes \delta\bar{q}(\delta\boldsymbol{\theta}), \qquad \delta\bar{q}(\delta\boldsymbol{\theta}) \approx [1,\ \delta\boldsymbol{\theta}^\top/2]^\top} \tag{109}$$

[5] $\boldsymbol{f}$ is the same dynamics function as in Section 4, while h is kept non-bold to avoid conflict with the angular momentum $\boldsymbol{h}$ from Section 3

[6] Without reset, the error state would grow and the first-order linearization in Eq. (111) would degrade.

for small $\delta\boldsymbol{\theta}$ (exponential map $\exp : \mathfrak{so}(3) \to \mathrm{SO}(3)$ in quaternion form). In the **predict step**, the nominal state is propagated exactly without linearization. Given IMU measurements, where accelerometer $\boldsymbol{a}_m$, gyroscope $\boldsymbol{\omega}_m$, at time step Δt, with bias-compensated quantities $\hat{\boldsymbol{a}} = \boldsymbol{R}_{IB}(\hat{\bar{q}})(\boldsymbol{a}_m - \hat{\boldsymbol{b}}_a) + \boldsymbol{g}$, $\hat{\boldsymbol{\omega}} = \boldsymbol{\omega}_m - \hat{\boldsymbol{b}}_g$:

$$\boxed{\hat{\boldsymbol{r}}^+ = \hat{\boldsymbol{r}} + \hat{\boldsymbol{v}}\Delta t + \tfrac{1}{2}\hat{\boldsymbol{a}}\Delta t^2, \quad \hat{\boldsymbol{v}}^+ = \hat{\boldsymbol{v}} + \hat{\boldsymbol{a}}\Delta t, \quad \hat{\bar{q}}^+ = \hat{\bar{q}} \otimes [\cos(\tfrac{\|\hat{\boldsymbol{\omega}}\|\Delta t}{2}),\ \sin(\tfrac{\|\hat{\boldsymbol{\omega}}\|\Delta t}{2})\,\hat{\boldsymbol{\omega}}^\top/\|\hat{\boldsymbol{\omega}}\|]^\top} \tag{110}$$

Biases are constant in prediction. Meanwhile, the **error-state dynamics** evolve to first order as $\delta\boldsymbol{x}_{k+1} = \boldsymbol{F}_k\,\delta\boldsymbol{x}_k + \boldsymbol{w}_k$, where the transition matrix $\boldsymbol{F} \in \mathbb{R}^{15\times 15}$ has the block structure:

$$\boldsymbol{F} = \begin{pmatrix} \boldsymbol{I}_3 & \boldsymbol{I}_3\Delta t & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_3 & -\boldsymbol{R}\left[\boldsymbol{a}_m - \hat{\boldsymbol{b}}_a\right]_\times \Delta t & -\boldsymbol{R}\Delta t & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_3 - [\hat{\boldsymbol{\omega}}]_\times \Delta t & \boldsymbol{0} & -\boldsymbol{I}_3\Delta t \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_3 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_3 \end{pmatrix} \tag{111}$$

where $\boldsymbol{R} \equiv \boldsymbol{R}_{IB}$ is the nominal body-to-world rotation. The key cross-couplings are: orientation error misrotates acceleration ($\delta\boldsymbol{v} \leftarrow \delta\boldsymbol{\theta}$), accelerometer bias corrupts velocity ($\delta\boldsymbol{v} \leftarrow \delta\boldsymbol{b}_a$), and gyroscope bias corrupts orientation ($\delta\boldsymbol{\theta} \leftarrow \delta\boldsymbol{b}_g$). Since the error-state mean is zero by construction, only the **covariance** is propagated:

$$\boldsymbol{P}_k^- = \boldsymbol{F}_{k-1}\,\boldsymbol{P}_{k-1}\,\boldsymbol{F}_{k-1}^\top + \boldsymbol{Q}_{k-1} \tag{112}$$

where $\boldsymbol{Q}_k$ is the discrete-time process noise covariance, assembled from accelerometer noise, gyroscope noise, and bias random walk variances. In the **update step**, the innovation is $\boldsymbol{z}_k - h(\boldsymbol{x}_{\mathrm{nom},k})$ (nonlinear h evaluated on the nominal — no linearization for the mean), and $\boldsymbol{H}_k = \frac{\partial h}{\partial \delta\boldsymbol{x}}\big|_{\delta\boldsymbol{x}=\boldsymbol{0}}$ is the Jacobian w.r.t. the *error state* (not the full state):

$$\boxed{\boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}_k^\top(\boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^\top + \boldsymbol{N}_k)^{-1}, \quad \delta\hat{\boldsymbol{x}}_k = \boldsymbol{K}_k(\boldsymbol{z}_k - h(\boldsymbol{x}_{\mathrm{nom},k})), \quad \boldsymbol{P}_k = (\boldsymbol{I}_{15} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_k^-} \tag{113}$$

After the update, the **reset step** injects $\delta\hat{\boldsymbol{x}}$ into the nominal state via $\boxplus$ (additive for $\boldsymbol{r}, \boldsymbol{v}, \boldsymbol{b}_a, \boldsymbol{b}_g$; $\hat{\bar{q}} \leftarrow \hat{\bar{q}} \otimes \delta\bar{q}(\delta\hat{\boldsymbol{\theta}})$ for orientation) and resets $\delta\hat{\boldsymbol{x}} \leftarrow \boldsymbol{0}$, $\boldsymbol{P} \leftarrow \boldsymbol{G}\boldsymbol{P}\boldsymbol{G}^\top$ where $\boldsymbol{G} \approx \boldsymbol{I}_{15}$ [7]. The following table summarizes the **differences between the standard EKF and the ESKF**:

	Standard EKF	**ESKF**
State	Full state (may include quaternion)	Nominal + error in tangent space
Orientation error	Additive (violates constraints)	Multiplicative (respects SO(3))
Linearization point	Current estimate (can be far from truth)	Always near $\delta\boldsymbol{x} = \boldsymbol{0}$
Mean propagation	Linearized	Exact (nonlinear integration)
Covariance size	16×16 (with quaternion)	15×15 (minimal)
Singularity-free	No (Euler) / Constrained (quaternion)	Yes

5.3 IMU-Kinematic State Estimation for Legged Robots

This section applies the ESKF from Section 5.2 to legged robots, fusing IMU data with joint-encoder measurements through the forward kinematics and Jacobians from Sections 2–3.

The **IMU measurement model** provides body-frame specific force and angular velocity:

$$\boldsymbol{a}_m = \boldsymbol{R}_{IB}^\top(\boldsymbol{a} - \boldsymbol{g}) + \boldsymbol{b}_a + \boldsymbol{n}_a, \qquad \boldsymbol{\omega}_m = {}_B\boldsymbol{\omega}_{IB} + \boldsymbol{b}_g + \boldsymbol{n}_g \tag{114}$$

with $\boldsymbol{n}_a \sim \mathcal{N}(\boldsymbol{0}, \sigma_a^2\boldsymbol{I}_3)$, $\boldsymbol{n}_g \sim \mathcal{N}(\boldsymbol{0}, \sigma_g^2\boldsymbol{I}_3)$, $\boldsymbol{a}$ true world-frame acceleration, $\boldsymbol{g} = [0, 0, -g]^\top$, and $\boldsymbol{R}_{IB}$ the body-to-world rotation matrix. Biases evolve as random walks: $\dot{\boldsymbol{b}}_a = \boldsymbol{n}_{ba}$, $\dot{\boldsymbol{b}}_g = \boldsymbol{n}_{bg}$. When foot i

[7] Exact for the "global" error convention; the "local" convention has a small $\left[\delta\hat{\boldsymbol{\theta}}\right]_\times$ correction, usually omitted.

is in stance, its world-frame position ${}_I\boldsymbol{r}_{\text{foot},i}$ is approximately constant between footfalls, so the base position can be expressed via **forward kinematics**:

$$\boxed{{}_I\boldsymbol{r}_B = {}_I\boldsymbol{r}_{\text{foot},i} - \boldsymbol{R}_{IB}\,{}_B\boldsymbol{r}_{\text{FK},i}(\boldsymbol{q}_j)} \tag{115}$$

where ${}_B\boldsymbol{r}_{\text{FK},i}(\boldsymbol{q}_j) \in \mathbb{R}^3$ is the foot position in the body frame, computed from joint angles $\boldsymbol{q}_j$ via the kinematic chain. Differentiating Eq. (115) under the stance assumption ${}_I\dot{\boldsymbol{r}}_{\text{foot},i} = \boldsymbol{0}$ gives the **velocity measurement**:

$${}_I\boldsymbol{v}_B = -\boldsymbol{R}_{IB}\,\boldsymbol{J}_{\text{foot},i}(\boldsymbol{q}_j)\,\dot{\boldsymbol{q}}_j - {}_I\boldsymbol{\omega}_{IB} \times \boldsymbol{R}_{IB}\,{}_B\boldsymbol{r}_{\text{FK},i}(\boldsymbol{q}_j) \tag{116}$$

where $\boldsymbol{J}_{\text{foot},i}$ is the foot Jacobian mapping joint velocities to body-frame foot velocity. The **observation Jacobian** $\boldsymbol{H}_k$ in Eq. (113) is taken with respect to the error state. When foot positions ${}_I\boldsymbol{r}_{\text{foot},i}$ are augmented into the state, the observation model becomes the kinematic constraint $h(\boldsymbol{x}) = {}_I\boldsymbol{r}_{\text{foot},i} - {}_I\boldsymbol{r}_B - \boldsymbol{R}_{IB}\,{}_B\boldsymbol{r}_{\text{FK},i}$, which should be zero in perfect stance. The resulting Jacobian blocks for the position measurement are: $\frac{\partial h}{\partial \delta \boldsymbol{r}} = -\boldsymbol{I}_3$ (position enters with negative sign), $\frac{\partial h}{\partial \delta \boldsymbol{\theta}} = \boldsymbol{R}_{IB}\,[{}_B\boldsymbol{r}_{\text{FK},i}]_\times$ (orientation error rotates the leg vector), $\frac{\partial h}{\partial \delta \boldsymbol{r}_{\text{foot},i}} = +\boldsymbol{I}_3$ (foot position enters with positive sign), and all other blocks are zero. The kinematic update is only applied when the foot is in stance. **Contact detection** methods include:

- **Force/torque sensors:** stance if $f_z > f_{\text{thresh}}$.
- **Gait phase:** known stance/swing schedule from the controller.
- **Probabilistic:** weight the measurement noise covariance $\boldsymbol{N}_k$ by contact probability (high $\boldsymbol{N}_k$ during swing effectively ignores the measurement).

Not all states are **observable** from IMU and leg kinematics alone:

State	Observable?	Source
Roll, pitch	Yes	Gravity vector in accelerometer
Yaw	No	No absolute heading reference
Vertical position (z)	Yes	Leg kinematics (stance height)
Horizontal position (x, y)	Relative only	Observable w.r.t. stance foot; drifts absolutely
Velocity	Yes	IMU integration + kinematic corrections
IMU biases	Partially	Observable through kinematic measurements

Yaw and horizontal position drift over time [8]. In practice, the following considerations apply:

- **Multi-rate fusion:** Predict at IMU rate (200–1000 Hz), kinematic update at encoder rate (100–500 Hz).
- **Foot slip:** Inflate $\boldsymbol{N}_k$ during suspected slip or use robust norms, e.g., Huber, on innovations.
- **Initialization:** Standing still: roll/pitch from gravity, position/velocity from kinematics, biases zero with large $\boldsymbol{P}_0$.

[8] This is a fundamental proprioceptive limitation. Bounding drift requires exteroceptive sensors (camera, LiDAR) for absolute or loop-closure measurements.

Appendix

Quick Reference: Key Equations

Kinematics

Change of frame	$_A\boldsymbol{r} = \boldsymbol{R}_{AB}\,{}_B\boldsymbol{r}$
Rotation matrix derivative	$\dot{\boldsymbol{R}}_{AB} = [{}_A\boldsymbol{\omega}_{AB}]_\times \boldsymbol{R}_{AB} = \boldsymbol{R}_{AB}\,[{}_B\boldsymbol{\omega}_{AB}]_\times$
Rodrigues' formula	$\boldsymbol{R} = \boldsymbol{I} + \sin\theta\,[\hat{\boldsymbol{n}}]_\times + (1-\cos\theta)\,[\hat{\boldsymbol{n}}]_\times^2$
Homogeneous transform	$\boldsymbol{T}_{AB} = \begin{bmatrix} \boldsymbol{R}_{AB} & \boldsymbol{r}_{AB} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}, \quad \boldsymbol{T}_{AB}^{-1} = \begin{bmatrix} \boldsymbol{R}^\top & -\boldsymbol{R}^\top\boldsymbol{r} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}$
Transport equation	$\boldsymbol{v}_P = \boldsymbol{v}_B + \boldsymbol{\omega} \times \boldsymbol{r}_{BP}$
Acceleration	$\boldsymbol{a}_P = \boldsymbol{a}_B + \boldsymbol{\alpha} \times \boldsymbol{r}_{BP} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}_{BP})$
Forward kinematics	$\boldsymbol{T}_{0n} = \prod_{i=1}^{n} \boldsymbol{T}_{(i-1)i}(q_i)$
Geometric Jacobian	$[\boldsymbol{v}; \boldsymbol{\omega}] = \boldsymbol{J}_G\,\dot{\boldsymbol{q}}$
Pseudoinverse (redundant)	$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}\dot{\boldsymbol{x}}_d + (\boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J})\dot{\boldsymbol{q}}_0$
Damped least squares	$\dot{\boldsymbol{q}} = \boldsymbol{J}^\top(\boldsymbol{J}\boldsymbol{J}^\top + \lambda^2\boldsymbol{I})^{-1}\dot{\boldsymbol{x}}_d$

Dynamics

Newton's equation	$\boldsymbol{f} = m\,\boldsymbol{a}_c$
Euler's equation	$\boldsymbol{\tau} = \boldsymbol{\Theta}\,\boldsymbol{\alpha} + \boldsymbol{\omega} \times (\boldsymbol{\Theta}\,\boldsymbol{\omega})$
Parallel axis theorem	$\boldsymbol{\Theta}_O = \boldsymbol{\Theta}_C - m\,[\boldsymbol{r}_{OC}]_\times^2$
Mass matrix	$\boldsymbol{M} = \sum_i (m_i \boldsymbol{J}_{v_i}^\top \boldsymbol{J}_{v_i} + \boldsymbol{J}_{\omega_i}^\top \boldsymbol{\Theta}_i \boldsymbol{J}_{\omega_i})$
Kinetic energy	$T = \frac{1}{2}\dot{\boldsymbol{q}}^\top \boldsymbol{M}\dot{\boldsymbol{q}}$
Euler–Lagrange	$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} - \frac{\partial L}{\partial q_j} = \tau_j$
Equations of motion	$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{S}^\top\boldsymbol{\tau}$
Christoffel symbols	$c_{ijk} = \frac{1}{2}(\partial_k M_{ij} + \partial_j M_{ik} - \partial_i M_{jk})$
Skew-sym. property	$\dot{\boldsymbol{M}} - 2\boldsymbol{C}$ is skew-symmetric
Floating base	$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{b} + \boldsymbol{g} = \boldsymbol{S}^\top\boldsymbol{\tau}_j + \boldsymbol{J}_c^\top\boldsymbol{\lambda}$

Control

PD + gravity comp.	$\boldsymbol{\tau} = \boldsymbol{K}_p\boldsymbol{e} - \boldsymbol{K}_d\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q})$
Computed torque	$\boldsymbol{\tau} = \boldsymbol{M}(\ddot{\boldsymbol{q}}_d + \boldsymbol{K}_d\dot{\boldsymbol{e}} + \boldsymbol{K}_p\boldsymbol{e}) + \boldsymbol{b} + \boldsymbol{g}$
Impedance control	$\boldsymbol{\tau} = \boldsymbol{J}^\top(\boldsymbol{K}_p\boldsymbol{e}_x - \boldsymbol{D}\dot{\boldsymbol{x}}) + \boldsymbol{g}(\boldsymbol{q})$
Task-space inertia	$\boldsymbol{\Lambda} = (\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^\top)^{-1}$
Task-space dynamics	$\boldsymbol{\Lambda}\ddot{\boldsymbol{x}} + \boldsymbol{\mu} + \boldsymbol{p} = \boldsymbol{f}$
Task-space inv. dyn.	$\boldsymbol{f} = \boldsymbol{\Lambda}(\ddot{\boldsymbol{x}}_d + \boldsymbol{K}_d\dot{\boldsymbol{e}}_x + \boldsymbol{K}_p\boldsymbol{e}_x) + \boldsymbol{\mu} + \boldsymbol{p}$
Dyn. consistent $\boldsymbol{J}^{+}$	$\bar{\boldsymbol{J}} = \boldsymbol{M}^{-1}\boldsymbol{J}^\top\boldsymbol{\Lambda}$

Disclaimer

This summary condenses the original RSL Robot Dynamics lecture notes ($\sim$ 100 pages). The following topics from the original notes are **omitted or significantly reduced**: **Kinematics (Ch. 2 of original):**

- Unit quaternion representation and quaternion time derivatives (Sec. 2.4.5, 2.5)
- Active vs. passive rotation distinction (Sec. 2.4.2)
- Cylindrical and spherical coordinate systems for position and velocity (Sec. 2.2, 2.3)
- Time derivatives of non-ZYX Euler angle conventions (XYZ, ZYZ, ZXZ) (Sec. 2.5)
- Trajectory control for position and orientation (Sec. 2.9.4)

Dynamics (Ch. 3 of original):

- Constraint consistent dynamics—support null-space projector $\boldsymbol{N}_c$ derivation (Sec. 3.7.2)
- Inverse dynamics for floating-base systems as optimization—hierarchical least-square problems, iterative null-space projection, sequence of constrained QPs, task-space control as QP (Sec. 3.10)
- Quasi-static (virtual model) control for floating-base systems (Sec. 3.11)
- Operational space control with hybrid motion/force selection matrices $\boldsymbol{S}_M$, $\boldsymbol{S}_F$ (Sec. 3.9.4)

Appendices and general:

- Appendix A: Matlab code for multi-task control and IK for rotations examples
- Appendix B: Matlab code for 3D rotations (Euler angles $\leftrightarrow$ rotation matrices)
- Detailed step-by-step derivations and proofs
- Worked examples with full solutions
- Figures, diagrams, and bibliography

For the complete treatment, refer to the original lecture notes.